Quantitative continuity and computable analysis in Coq*

Florian Steinberg[1], Laurent Théry[2], and Holger Thies[3]

[1]INRIA, Saclay [2]INRIA, Sophia Antipolis [3]Kyushu University, Fukuoka

Abstract

We give a number of formal proofs of theorems from the field of computable analysis. Many of our results specify executable algorithms that work on infinite inputs by means of operating on finite approximations and are proven correct in the sense of computable analysis. The development is done in the proof assistant Coq and heavily relies on the Incone library for information theoretic continuity. This library is developed by one of the authors and the paper can be used as an introduction to the library as it describes many of its most important features in detail. While the ability to have full executability in a formal development of mathematical statements about real numbers and the like is not a feature that is unique to the Incone library, its original contribution is to adhere to the conventions of computable analysis to provide a general purpose interface for algorithmic reasoning on continuous structures.

The results that provide complete computational content include that the algebraic operations and the efficient limit operator on the reals are computable, that certain countably infinite products are isomorphic to spaces of functions, compatibility of the enumeration representation of subsets of natural numbers with the abstract definition of the space of open subsets of the natural numbers, and that continuous realizability implies sequential continuity. We also describe many non-computational results that support the correctness of our definitions. These include that the information theoretic notion of continuity used in the library is equivalent to the metric notion of continuity on Baire space, a complete comparison of the different concepts of continuity that arise from metric and represented-space structures and the discontinuity of the unrestricted limit operator on the real numbers and the task of selecting an element of a closed subset of the natural numbers.

The paper briefly describes Incone's sub libraries mf and Metric which may be of separate interest and have fewer dependencies and can thus be acquired separately. We occasionally mention additional material from the sister library CoqRep that contains more experimental concepts in attempt to more conveniently manipulate algorithms on infinite data inside of Coq while avoiding a full formalization of a model of computation.

*This work was supported by JSPS KAKENHI Grant Number JP18J10407, by the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks), by the ANR project FastRelax (ANR-14-CE25-0018-01) of the French National Agency for Research and by EU-MSCA-RISE project 731143 "Computing with Infinite Data" (CID).

Contents

1 Introduction

Computable analysis is the theory of computing on continuous structures. Its roots are often cited as going back to the very 1936 paper in which Turing introduced his machines [Tur36]. Turing's original definitions rely on the binary representation and he adapted them to the ones still used today in his 1937 correction [Tur38] with a pointer to earlier work by Brouwer. The theory of computable functions on the real numbers was further developed in the 1950s by Grzegorczyk and Lacombe in parallel [Grz57, Lac58]. Later on, Kreitz and Weihrauch extended the theory to apply to more general spaces and introduced the formal framework of representations that is standard today [KW85, Wei00, LN15]. The basic idea behind computable analysis is fairly easy to grasp: To make uncountable structures available to computation, one encodes them by infinitary objects that can still be operated on mechanically. Most commonly infinite strings are used, but more conveniently one may use functions between discrete structures: A reasonable encoding for real numbers is to describe them by functions that provide arbitrarily accurate approximations. Since the inputs and outputs of such functions can be chosen rational and thus be described by finite means, this leads to a realistic model of computation. To compute functions on the real numbers, one operates on these encodings and algorithms use a computation model that can handle properly infinite inputs while remaining realistic in the sense of being implementable. The standard references for more detailed information about computable analysis are [PER89, Ko91, Wei00] and the topics are presented in a form that is somewhat closer to how this paper proceeds in [Bau00, Sch02b, Pau16].

The model of computation used in computable analysis must be distinguished from more common ones that operate on functional inputs by encoding them via a Gödel numbering and, from the perspective of computable analysis, the latter can be understood as imposing a more relaxed notion of correctness of algorithms [AB14, LN15]. Neither of these methods of doing real number computation reflects the practices from numerical analysis. For the sake of efficiency, numerical analysts rely on the use of floating point computations in implementations of their algorithms. This is while proofs of correctness use mathematical methods whose underlying notion of an algorithms are geometric modes of computation similar to the BSS model [BSS89] that assume the capability to carry out exact operations on real numbers. In the implementation, real variables are substituted with machine numbers for which basic properties like associativity fail. This leads to a situation where the mathematical proof of correctness of the algorithm need not say anything about the correctness of an implementation even if both are done correctly. On one hand, these problems are well aware to algorithm designers and considered relevant in many applications that demand high reliability of the results. On the other hand they are difficult to overcome as by-hand error estimation of more complicated algorithms quickly becomes infeasible and is error-prone itself.

Recently, with the growing maturity of formal methods for software verification and proof assistants, a new approach to ensure the reliability of floating point computations has become accessible [BM17a]. Over the last decade an active community has formed whose focus is to apply formal methods to floating point algorithms. These efforts have a fairly large coverage of different topics reaching from verification of single algorithms [BCF$^+$13, BRT17] to the formalization of methods from numerical analysis [BCF$^+$17]. While mathematical development of numerical schemes and implementation are a priori very different tasks in a formal development they come hand in hand: The formalization of the numerical solution schemes has to precede an attempt to prove software that uses this scheme correct. There are a number of proof assistants that are appropriate for such endeavors and one that is particularly popular in this community is Coq [BLM16]. This is because the Coq system is traditionally centered around the interface between proofs and computation. Indeed, Coq uses a type-theoretic setting that favors constructive reasoning and allows for code extraction from proofs, but also provides advanced automation tools for instance for proving inequalities over real numbers. Furthermore, it provides a designated type `Prop` meant to distinguish proofs with computational content from those that are purely for specification and verification. The recent advances in formal proofs and verified numerics have decreased the gap between the theory of computation and numerical practice and the computable analysis community has shown an increase of interest in these developments [MPPZ16]. In part because the focus of computable analysis is reliability as all algorithms must provide rigorous error bounds, but also since algorithms from computable analysis are notoriously difficult to implement in a way that makes them competitive in terms of speed and memory consumption [Mül01, Bla01, KN17].

1.1 Coq and proofs about continuous structures and computation

Coq is a proof assistant that supports mathematicians in giving fully formal proofs of their results. A typical Coq development consists, just like a mathematical paper, mostly of definitions and lemmas (and some explanations and documentation). The definitions specify the objects that the developer is interested in and the lemmas about their properties. Coq automatically checks that definitions are well formed and that only correct proofs are specified for the lemmas. For the definition of a function, for instance, Coq checks that returnvalues are specified for all possible inputs. This means that only total functions can be defined in Coq by design and this is necessary for theoretical reasons, as Coq heavily relies on the Curry-Howard correspondence. Mathematically one can still reason about partial functions by modeling a partial function from S to T by a total function from S to $\mathrm{opt}\, T$, where $\mathrm{opt}\, T$ is the extension of T with a single object that is understood to stand for "undefined". Proving a fully constructive result in Coq can be seen as using a high level programming language to specify how a desired result can be obtained from the given inputs. Some of the inputs here may not be traditional input data but instead evidence that the assumptions of the lemma are fulfilled. In principle there is not much difference between such a lemma and a definition of a function. One may even follow a Coq-definition with a proof (that should end in "Defined" instead of "Qed"), providing missing parts in the high level language instead of specifying them by hand.

For code extraction to be possible one needs to restrict to constructive reasoning. A mathematician working with Coq will quickly run into statements that appear to be true but that he can not seem to be able to prove. An example for a restriction of the internal logic of Coq that often causes troubles with mathematicians is functional extensionality: For functions f and g of the same type mathematicians would assume that the statement `(forall a, f(a) = g(a)) -> f = g` is true, but this is not provable in Coq. However, Coq allows to assume axioms and one may state additional inference rules as such. Many mathematical developments force the truth of functional extensionality by assuming it as an axiom and another popular axiom is classical reasoning or Markov's Principle. Of course, one has to make sure that the axioms are compatible with Coq's internal logic and compatible with each other. Coq's official webpages list some known facts about consistencies of axioms that are often used `https://github.com/coq/coq/wiki/The-Logic-of-Coq#axioms`. One should also be aware that assuming axioms impedes the ability to extract algorithmic information from proofs.

Even if all reasoning is constructive, many lemmas during a development will have parts whose computational content is only to provide evidence that some correctness statement is fulfilled. This may be that a definition of a function the developer is actually interested in fulfills a specification, i.e. that an algorithm is correct. Typically, if the correct specification was proven, the rest of the development will not rely on specifics of this proof either, so that it can be given a name, marked as correct and its details hidden. Indeed, this is important to allow Coq to efficiently check the later proofs in a big development, i.e. to keep proof terms manageable in size. In Coq there exists a type `Prop` that can, and is meant to, be used to mark parts of proofs that do not have computational content in the above sense. The distinguishing feature of this type is that a definition of a function cannot depend on the details of the proofs of inputs of type `Prop`. This rule is what allows the code extraction machinery of the Coq system to disregard all parts that are propositional.

Let's say that we extract an algorithm from a function with a regular input and an additional propositional input that asserts that the regular input has some property. The computational content of such a function can be thought of as an algorithm that only takes the regular input. The propositional input that is supposed to assert a property of the regular input was removed in the extraction of computational content and the user is assumed to make sure that he only uses the algorithm on valid inputs and is only guaranteed to get a meaningful result in this case. The propagation of propositional correctness can be considered verifying such algorithms. In Coq this propagation is usually done by specification lemmas and included in the development with other lemmas that have computational content. From a practiacl perspective, it is reasonable to treat the proofs of the specification lemmas differently. As these do not carry computational content, the rules for proving them can be adapted by assuming appropriate axioms: A user might take the stance that it is enough to be sure that no counter-example for correctness can be explicitly given. In this case a specification lemma is justified to use classical reasoning and assume any set of axioms that the user is convinced are consistent. This can vastly increase convenience in verification of algorithms. In cases where it does not cause extensive extra work or impact the comprehensibility of statements, it is still reasonable to keep specification lemmas as constructive as possible to be able to extract information in case it becomes useful.

The Coq system supports its users in keeping the distinction between computational and non-computational content up and prevents them from using non-computational content definitionally. Many of the axioms from the Coq system are formulated propositionally so that the user is prompted for incorrect use. This is for instance true for the choice principles in the standard library. The mechanism is not foolproof but generally works well. Unfortunately, the same mechanism appears as a major hurdle to user who do classical mathematics, where more liberal definitional thinking is common practice. Consider the function $\mathtt{up}\colon \mathbb{R} \to \mathbb{Z}$ that is part of the axiomatization of the reals in standard library and is supposed to return the least integer bigger than its input. The existence of such a function cannot be proven constructively in reasonable constructive instantiations of the real numbers. The existence of this function is still stated definitionally and not hidden behind an existential quantifier to make it a proposition. The motivation behind this is clear: it avoids many instances of having to resolve an existential quantifier and uses of a uniqueness lemma. I.e., Coq will no longer attempt to forbid the use of this function in definitions. A maybe even more prominent example is the boolean-valued inequality relation that allows branching over inequalities of reals. As a consequence, the use of Coq's code-extraction is very limited for obtaining algorithmic information from statements about the real numbers in the standard library. At best, geometric algorithms can be extracted and a replacement of the real number type by a realistically implementable one leads to an almost guaranteed loss of correctness.

Nonetheless, there exists a vast body of work building on the classical axiomatization of the reals in the standard library. For instance the Coquelicot library [BLM15] as a widely used library for real analysis that is conservative over this axiomatization. More recent developments in Coq's community for formalization of results from analysis take an even clearer stance on these topics and assume the full strength ε-axiom to marry the Coquelicot library with the mathematical components library [ACR18]. Computational content is restored in an additional step by using the mathematical libraries to prove floating-point algorithms correct [BM17b]. Computable analysis fits well into these developments: it traditionally uses classical reasoning on the mathematical structures and the algorithmic content is considered extra information about data representation that should follow the mathematical understanding.

The Incone library is an attempt to implement computable analysis and its backwards approach in Coq and use it and synergy with the developments in the verified numerics community to complement the forwards approach of working completely constructive. It stays faithful to Coq's propositions but additionally uses an internal construction that achieves a similar goal in a different context. For this it relies on the Rlzrs library that reflects some concepts similar to the mechanisms behind the code extraction and investigates them as a mathematical construction in Coq. The main reason for this reflection is that computable analysis is the piece of mathematics that we want to formally reason about. We want it to be possible to specify algorithms on a level of abstraction as this is usually done in computable analysis. However, another welcome consequence is that we obtain a separation into a mathematical layer and a computational layer on each of which the appropriate tools can be used. Currently our most used tool on the computational level is the Mathematical Components library and for mathematics it is the standard library and Coquelicot in combination with the Rstruct file. For mathematical analysis we hope to soon be able to move to mathcomp analysis. On the computational level use of coq-Inverval seems promissing in the future.

The reflection using the RLZRS library leads to a loss of the ability to extract algorithms from fully constructive proofs about the mathematical structures. However, some of these capabilities can be recovered, for instance, by proving induction principles for represented spaces. The use of such induction principles in computable analysis becomes subtle if computability is taken into consideration. This is related to uniformity issues and to the failure of the category of represented spaces with computable functions to have countable products, which is one of the most commonly encountered problems in obtaining computability versions of continuity results. Proofs of some restricted induction principles and some applications can be found in INCONE's sister library COQREP. However, in our experience, general purpose induction principles are prone for being unreasonably inefficient on this level and for this reason we have not attempted to provide any for INCONE. In concrete cases a better idea is to try to substitute them with induction principles on the level of discrete data by a change of representation or design the corresponding algorithms by hand.

The C-CoRn library for constructive analysis is by far the most advanced fully computational COQ development that deals with real numbers [CFGW04]. It achieves the executability by restricting to constructive proofs and the relation of our work to this should be clear from the previous paragraphs. The C-CoRn library provides a wide range of results about functions on real numbers and some about operators on function spaces. It includes an exhaustive treatment of metric spaces and uniformly continuous functions between metric spaces [O'C09]. The C-CoRn library is inspired by, and roughly follows the development of constructive analysis by Bishop and Bridges [BB12]. Our treatment of the real numbers rarely goes beyond what is already content of C-CoRn and many parts are inspired by it. This said, it should also be noted that the constructive nature may make the C-CoRn library and the publications related to it difficult to access for some classically trained mathematicians. Some of our results about reals are also covered by a smaller project that implemented Cauchy reals to use them and the mathematical components library to give a definition of the algebraic real numbers in COQ [Coh12]. While our treatment of computation on the real numbers rarely leaves the shadow other developments, in particular the C-CoRn library, some of the results about metric spaces do. To the best of our knowledge most of the rest falls outside of the scope of any other formal development in COQ, or in other proof assistants for that matter.

None of our results are mathematically original, but all formalize well-known facts from computable analysis. In our opinion this is an instance where many parts of the formalization itself are creative contributions. This is reflected in our presentation of the contents that regularly diverges from the standard approaches. We feel that most of these deviations are beneficial for the understandability and some are improvements due to new insights we gained through the formalization. The applications presented in this paper are pure computable analysis and the specified algorithms are far from being competitive. We currently use rational numbers for approximating reals, no kind of efficiency can be expected before these are not at least replaced by arbitrary precision floating-point numbers. However, it should be kept in mind that this is possible in principle and we believe the framework we use to have realistic applications. Indeed, the long term goal behind the development of the INCONE library is to provide an environment in which the intersection of formal proofs, computable and numerical analysis can conveniently be investigated in COQ and their merits can be combined in attempts to prove efficient algorithms with practical relevance correct.

1.2 Realizability approach to computation on finite and infinite data

Fix some set D of data, some set X of abstract objects and a relation on $D \times X$ that specifies which pieces of data describe (or approximate) which abstract objects. For now let us call such a relation a realizability relation if each abstract object is described by at least one piece of data. Realizability relations are ubiquitous in the theory of computation, constructive mathematics and proof theory. Depending on the field, realizability relations may be interpreted as a specification of a function in either direction and, depending on this choice, the condition that is imposed on a realizability relation can be formulated as being surjective or total. A priori, a realizability relation does not have a preferred direction and some fields even decide to omit the abstract objects completely and only talk about partial equivalence relations on the set of data.

The convention in computable analysis is to interpret a realizability relation as a specification of a function from data to abstract objects and to pick Baire space as the space of data. This leads to the notion of a multi-representation where a piece of data gives a description of an abstract object by means of providing on demand information about it. The description of real numbers via functions that take rational accuracy requirements and return rational approximations is an example. As in this case, it is most common that each of these descriptions uniquely identifies the abstract object, and in this case the relation is called single-valued. Any single-valued realizability relation can be identified with a partial surjective function that is then called a representation and is the central object of computable analysis. The INCONE library follows these ideas from computable analysis closely to provide a formal definition of represented spaces in COQ. However, as implicitly done above, it adds an additional layer of abstraction, where the inputs and outputs of a description need not always be explicitly encoded as natural numbers but are allowed to use any countable and inhabited types. If COQ's types are interpreted as sets and a classical setting is assumed, computable analysis is recovered. In a constructive setting or if one wants to reason about computability as refinement of continuity, more care has to be taken with the input and output types. One way to ensure that everything works out fine is to guarantee that the types are either finite or there is an effective bijection with the natural numbers. This may be forced by requiring the construction of Mathematical Components `countType` structure for the input and output types.

An opposing view can be useful in other applications. One may take the fact that any abstract object is hit by the relation as indication that a choice function through the relation that goes from abstract objects to data is an important concept and we call such a function a "representative function". Such a function is for instance useful for the sake of making quotients usable in COQ [Coh13] and particularly often encountered in the Mathematical Components library [mat]. Consider, for instance, the case of rational numbers. Here, the data would be pairs of integers, the abstract objects would be rational numbers, which may be thought of as equivalence classes of pairs of integers. The representative function could in this case pick from an equivalence class the unique fully canceled pair of integers. A multitude of further examples can be found in the Mathematical Components library. Due to the change in directions, where computable analysis restricted to single-valued representations, representative functions are often required to be injective. Indeed, in their above use for quotients this can be seen as a consequence of the definitions.

From the perspective of computable analysis the concept of an representative function is rarely useful. Picking a unique reference description from the continuous Baire space is highly ineffective for most representations that do not allow reformulation using a discrete data set. The difference in applicability of this concept between computable analysis and the Mathematical Components library can be attributed to the difference in scope: The Mathematical Components library is mostly concerned with operating on objects that can be encoded by finite means efficiently while in computable analysis the focus of interest is on operating on objects from sets of continuum cardinality where this is not possible anymore. Indeed, the reader may go through the concretely specified representations from this paper and verify that a computable representative function exists exactly for the discrete example spaces.

Attempts to give a computability theoretic treatment of INCONE's input and output types such that stability under change of the level of abstraction can be guaranteed seem to lead back to concepts very similar to representative functions. This is far from a rigorous argument but may explain our empirical observation that the types from the Mathematical Components library are often a good fit for use as input and output types for descriptions in our examples. The reason why the INCONE library does not globally require the input and output types to be `countTypes` is for the sake of providing a better interface with data-types outside of the math-comp ecosystem. While the Mathematical Components library is very efficient in preserving its structures through the most common type-constructions, it can be tedious to construct new instances for custom data-types. For example, in our development the rational numbers from the standard library were preferred over the corresponding math-comp type because they provide a better interface with the reals from the standard library. Proving the rationals from the standard library to be countable was a matter of minutes. The construction of an appropriate `countType` would have been possible, but it would have been a considerably bigger effort at least for someone not native to the math-comp system.

The formulation of concepts from computable analysis in the INCONE library relies on the RLZRS library, which in turn is based on the MF library for manipulation of multivalued functions. Multifunctions are very popular for specification and classification of problems in computable analysis [BKMP16, BGP17, BG11, BDBP12, PS18]. Within this field, multifunctions form a topic of research of their own [Pau12, PZ13]. This is not to say that this concept was invented for computable analysis, multifunctions have been popular in other branches for a long time. For instance in computational complexity, in particular the theory of promise problems and non-deterministic computation [Sel94, ASBZ13], and even in the treatment of non-smooth and non-linear problems in functional analysis [EM46, Dei92].

Due to their many potential applications outside of the INCONE library, the development of a convenient environment for manipulation of multifunctions was exported and can be obtained separately as the MF-library [Ste19d]. Already the INCONE library uses multifunctions for several different purposes: Through the RLZRS library for the formulation of realizability, but for instance also for dealing with partiality issues in coq. Our last example even features mulitfunctions in the role they traditionally play in computable analysis: A popular topic is to prove mathematical problems computable or, if this is impossible, classifying their degree of incomputability. Here, mathematical problems are formulated as multifunctions between represented spaces and the comparison is carried out by means of Weihrauch reductions. A class of examples of mathematical problems, or computational tasks that often appear in such classifications is closed choice on some space $\mathbf{X}$, where the task is "given a non-empty closed set $A \in \mathbf{A}(\mathbf{X})$ select an element $a \in A$". A closed subset of a represented space is given by specifying positive information about its complement. Thus, for most choices of $\mathbf{X}$, this task is uncomputable and even discontinuous. We give a formal proof of this statement for the special case $\mathbf{X} = \mathbb{N}$ that turns up in classifications especially often.

1.3 Outline of the paper and its relation to the Incone library

The paper describes the INCONE library in detail and hopefully it is possible to use it as an introduction or as a manual. We selectively mention the important notations and whenever the name of a concept in the paper diverges from its name in the library, we point this out. One major point where paper and the library diverge is that INCONE uses the phrase "continuity space" for what is referred to as "represented space" in this paper. This is because the library is derived from the COQREP library which attempts to talk about computability more directly. Since represented spaces are tied to computability theory, which the INCONE library avoids apart from on the meta-level, we decided to switch from represented space to continuity space. Detailed instructions for installation and for verifying the content of this paper can be found on the paper's project page `https://holgerthies.github.io/continuity` and in the references [Ste19d, Ste19c, Ste19e, Ste19b, Ste19a]. The treatment of abstract realizability, which may be useful in more general settings, has been exported to a separate library called Rlzrs. The Rlzrs library uses a somewhat different language and we refrain from describing it in detail here.

The main contributions of this paper are mostly listed in Section 4 with some exceptions that already pop up in Section 3. All theorems, propositions and lemmas in this paper have been formalized in Coq and have explicit pointers to their name in the Incone library. Many of the claims that are stated in the plain text, as corollaries or as examples are also supported by formal proofs and the references to the Incone library are put in brackets after the statement. The major milestones in the development of the Incone library were an appropriate formulation of continuity, the construction of a continuous universal, of finite and countable products, function spaces and a duality operator. The formal proofs that we consider the main contributions are that the countably infinite product of a space with itself is isomorphic to a space of functions (Theorem 28), that the algebraic operations and the efficient limit operator on the reals are computable (Examples 19 and 22), compatibility of the enumeration representation of subsets of natural numbers with the abstract definition of the space of open subsets of the natural numbers (Theorem 38), and that continuous realizability implies sequential continuity (Theorem 8). The previous results are fully algorithmic, but we also describe many non-computational theorems. These include numerous specification results for the constructions the Incone library (in particular Theorems 9, 13, 27 and Proposition 21), that the information theoretic notion of continuity used in the library is equivalent to the metric notion of continuity on Baire space (Theorem 32), a complete comparison of the different concepts of continuity that arise from metric and represented-space structures (Theorems 35 and 36) and the discontinuity of the unrestricted limit operator on the real numbers (Example 22) and the task of selecting an element of a closed subset of the natural numbers (Theorem 41).

In the Section 2 we introduce the concept of continuity of partial operators on Baire space. As a preparation for a proper treatment of partiality in Coq, the introduction describes Incone's sub library mf for specification of functions through relations. The first part discusses how we reflect computability of functions and operators in Coq. A relativization of this construction is one of the core concepts that is revisited many times throughout the rest of the paper. This construction also provides a very smooth transition to considerations about continuity. The second part gives an information theoretic description of continuity on Baire space and an overview over the formalization of this notion in the Incone library. The third part presents the universal that the Incone library uses to implement the function space construction from computable analysis. These three parts together cover most of what can be found in the `baire_space` folder of the Incone library.

Section 3 deals with the basic concepts from computable analysis, explains how they are realized in the library and introduces the real numbers as an example that is used through the rest of the section. The first part of the section explicitly describes how a few of the simple type constructions like products are automatized in the library. It presents some examples that use these constructions to prove the algebraic operations on the real numbers and polynomial evaluation computable. The second part describes how countably infinite products can be constructed and considers point-wise operations on spaces of sequences and the limit operator on the real numbers as concrete examples. From a category theoretical point of view, infinite products are of particular interest as their existence is only guaranteed in the case where all continuous functions are considered as morphisms and fail to exist if one restricts to computable ones. The final third part builds exponentials using a construction that is known to work for both these categories. It presents a formal proof that the countably infinite products can be recovered as certain exponentials. As a whole the Section roughly corresponds to the content of the folder `continuity_spaces` in the Incone repository.

The final section (Section 4) starts with a brief description of the metric library and a comparison to other formalizations that have a similar purpose. The first part presents a formal proof that information theoretic notion of continuity that the Incone library uses internally is equivalent to the more traditional approach of equipping Baire space with an appropriate metric. The second part presents formal proofs about the relation of different concepts of continuity. The final part introduces Sierpiński space as a space that can be used to abstractly reason about open and closed subsets of represented spaces. It then shows that certain enumeration representations are a concrete instantiation of the structure that the power set of the natural numbers can be given when its elements are interpreted as the open or closed subsets. It uses this to give a formal proof that the task of closed choice on the naturals is discontinuous, i.e., that the multivalued function that corresponds to this task does not have a continuous realizer. The Weihrauch-degree that corresponds to this task is called $C_{\mathbb{N}}$ and is very commonly encountered in classifications of the computational content of mathematical theorems.

2 Multifunctions and partial operators on Baire space

Computable analysis transfers the computability and topological structure of Baire space to more general spaces by means of encodings that are called representations. Before we go into detail about how this can be done, this chapter describes the structure on Baire space that we need. Classically, Baire space is the space of all total functions from natural numbers to natural numbers, i.e., functions of type $\mathbb{N} \to \mathbb{N}$. We use a more general setting and refer to any space of the form $\mathbf{Q} \to \mathbf{A}$ as Baire space if $\mathbf{Q}$ and $\mathbf{A}$ are countable and inhabited types. Classically these assumption imply that the types are either finite or bijectively related to the natural numbers. Of course, constructively this is far from true. Indeed, if computability considerations come in, that is, if the surjection whose existence is guaranteed by the countability is considered an encoding of the elements of the type, more care has to be taken. The critical reader may in the following replace any occurrence of $\mathbf{Q}, \mathbf{A}$ and their dashed variants by $\mathbb{N}$ and assume that the difference in naming is merely for convenience in type-checking. In the applications that we look at, these substitutions can always be carried out by hand. Readers not familiar with Coq and the treatment of functions in proof assistants may go further and replace any mentioning of "Coq functions without axioms" by "functions definable in Gödels system T" or in the absence of functional inputs even by "primitive recursive functions". This correspondence is well known to be imperfect but happens to work out in all examples that we look at.

In Coq functions are always total and to find an appropriate notion of partiality, which is important for a proper treatment of continuity, we first need to discuss how functions can be specified through relations. Throughout the whole paper we use a happy mix of type-theoretic, set theoretic and mathematical notation. In particular we identify subsets of a given type T with functions of type $T \to$ `Prop` and borrow the element-hood notation from set theory, i.e., we write $t \in T$ for $T(t)$ (the corresponding notation `_ \from _` in the library is unfortunately very unstable and often not printed). We also use the mathematical notation for subsets, subset inclusion and partial functions. Finally, we avoid the use of the colon for typing when referring to element-hood of Baire spaces. This is because of the confusing ambiguity in interpretation of function types. For our purposes it is more natural to consider elements of Baire-spaces as mathematical functions and not elements of a function type. This is also reflected in common applications of the functional extensionality axiom to elements of Baire space and in a regular use of choice principles on the level of Baire spaces.

A multivalued function $F \colon S \rightrightarrows T$ (notation `_ ->> _` in the library) is a function that assigns to each $s \colon S$ a possibly empty subset $F(s)$ of T. While this gives F the type $S \to T \to$ `Prop` or equivalently $S \times T \to$ `Prop` and one could identify F with a binary relation, the intuition behind a multivalued function is different as S is treated as input type and T as output type. The domain of a multifunction F is given by $\mathrm{dom}(F) := \{s \colon S \mid \exists t \colon T, t \in F(s)\}$ and for $s \in \mathrm{dom}(F)$ the set $F(s)$ should be interpreted as the set of eligible return values. A multivalued function is called **total** if its domain is all of S, and **single-valued** if each $F(s)$ has at most one element.

Any multivalued function can be considered a specification for functions: A function $f\colon S \to T$ fulfills the specification $F\colon S \rightrightarrows T$ if $s \in \mathrm{dom}(F) \implies f(s) \in F(s)$ for all $s\colon S$. In this case we say that f **is a choice for** F (`icf` in the library with notation `_ \is_choice_for _`). The operations on multivalued functions are chosen such that they behave well with the interpretation as specifications. For instance, the **composition** $F \circ G$ of two multivalued functions $G\colon R \rightrightarrows S$ and $F\colon S \rightrightarrows T$ is given by

$$F \circ G(r) := \{t\colon T \mid G(r) \subseteq \mathrm{dom}(F) \wedge \exists s, t \in F(s) \wedge s \in G(r)\}.$$

(notation `_ \o _` in the library.) This is an associative operation and the second half of the requirement, namely $F \circ_R G(r) := \{t\colon T \mid \exists s, t \in F(s) \wedge s \in G(r)\}$, is what is commonly used as composition for relations. The domain condition is a modifier that addresses the difference in interpretations and in particular leads to a loss of the symmetry under exchange of the input and output types.

There are two very straightforward ways to generate multifunctions from functions or partial functions. Namely, for a function $f\colon S \to T$ just use the specification $\mathtt{F2MF}f\colon S \rightrightarrows T$ that uniquely determines it, i.e. $\mathtt{F2MF}f(s) := \{t\colon T \mid t = f(s)\}$. Clearly, this multifunction is always total and single-valued (`F2MF_tot` and `F2MF_sing`) and assuming that T is not empty and an appropriate choice principle, each total single-valued multifunction arises in this way (`fun_spec`). This construction can easily be extended to partial functions, i.e., by assigning to $g\colon S \to \mathrm{opt}\, T$ the function

$$\mathtt{PF2MF}g(s) := \{t\colon T \mid g(s) = \mathrm{Some}\, t\},$$

which is still single-valued (`PF2MF_sing`) but need not be total anymore. Again, assuming some choice axioms, one can show that any single-valued multifunction arises from some partial function (`pfun_spec`). Due to the setting of this paper, where we are mostly interested in operators on function types, coding a partial function as a function to an option type is often unnatural as it may be understood to indicate that the domain of the function should be decidable. We therefore choose the mathematical notation $g\colon\subseteq S \to T$ for partial functions to avoid mentioning option types and in the Incone library they are usually treated as single-valued multifunctions right away.

The assignments `F2MF` and `PF2MF` are compatible with the multifunction composition:

Lemma 1 (`F2MF_comp` and `PF2MF_comp`) *Composition of functions translates to multifunction composition, i.e.* $\mathtt{F2MF}(f \circ g) = \mathtt{F2MF}f \circ \mathtt{F2MF}g$ *and* $\mathtt{PF2MF}(f \circ g) = \mathtt{PF2MF}f \circ \mathtt{PF2MF}g$.

Indeed in the previous Lemma the multifunction composition could have been replaced with the relational composition, as the two compositions coincide if the function that is applied last is total (`comp_rcmp`) resp. the first one is single-valued (`sing_comp`). Since the internal logic of Coq does not imply propositional extensionality, and we prefer not to assume it globally, the equality of multivalued functions is handled as equivalence relation in our implementations (`equiv` in the library with notation `_ =~= _`). To be able to still conveniently manipulate multivalued functions, the setoid-rewrite mechanism of Coq is used.

Note that in contrast to functions, any multifunction can be assigned a reverse multifunction where the input and output is simply switched. All properties of a multifunction have a co-version that requires the same property for the reverse multifunction. Many of the co-properties have nice characterizations for the special cases of functions. For instance, a function f is injective if and only if $\mathtt{F2MF}f$ is co-single-valued (`mfinv_inj_sing`). Readers familiar with representations will not be surprised that we list the following as a lemma:

Lemma 2 (`PF2MF_cotot`) *A partial function f is surjective if and only if* $\mathtt{PF2MF}f$ *is co-total.*

Another operation on multivalued functions that is important for our purposes is the notion of a tightening (`tight` in the library with notation `_ \tightens _`). For multifunctions $F, G\colon S \rightrightarrows T$ we say that F **tightens** G if it is more restrictive as a specification. That is, if

$$\mathrm{dom}(G) \subseteq \mathrm{dom}(F) \quad \text{and} \quad \forall s \in \mathrm{dom}(G), F(s) \subseteq G(s).$$

Indeed, under appropriate assumptions F tightens G if and only if being a choice for F implies being a choice for G (`icf_tight` and `tight_icf`). A function f is a choice for a multifunction F if and only if `F2MF`f tightens F (`icf_spec`) and if `PF2MF`f tightens F we say that f is **partial choice** for F. Many properties of multifunctions are proven in the library and we just pick an example:

Lemma 3 (`tight_comp`) *If F tightens F' and G tightens G' then $F \circ G$ tightens $F' \circ G'$.*

An exhaustive overview over the concepts and notations for multifunctions the MF library provides can be found in the preamble of the `mf.v` file.

2.1 Capturing the computable multifunctions and relativization

Another, more involved but for the purposes of this paper important construction produces a multifunction Φ_N of type $S \rightrightarrows T$ from a function N of type $\mathbb{N} \times S \to \operatorname{opt} T$ via

$$\Phi_N(s) := \{t \colon T \mid \exists n, N(n, s) = \operatorname{Some} t\}.$$

The idea behind this (also called the Kleene normal-form theorem [Soa78]) is that in the special case where $S = \mathbb{N} = T$ the specification of any partial computable function can be expressed in this way with N a primitive recursive function. This is in particular interesting to us since any primitive recursive function has a definition in Coq that is closed under the global context [O'C05]. Note that a priori, the multifunction Φ_N need neither be total nor single-valued. A single-valued tightening $\Phi_{N'}$ of Φ_N can be obtained by setting N' to on input n and s run N on inputs $(0, s), \ldots, (n, s)$ and return first value returned by N (a definition and formal proof of correctness can be found in the CoqRep library). This construction actually forces more than single-valuedness, as the functions it produces are all **monotone** in the sense that if they return something they neither ever go to another value nor back to None. Thus, under the reasonable assumption that any function that has an axiom free definition in Coq is computable, each specification that can be produced has a computable partial choice function.

The $\Phi.$ correspondence makes it possible to talk about computable functions in Coq at least on a meta-level. Since we are interested in specification of partial operators, we relativize the construction. First recall that we use the notations $\mathcal{B} = \mathbf{Q} \to \mathbf{A}$ and $\mathcal{B}' = \mathbf{Q}' \to \mathbf{A}'$. We consider a function $M : \mathbb{N} \times \mathcal{B} \times \mathbf{Q}' \to \operatorname{opt} \mathbf{A}'$ to specify the operator $F_M \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ such that

$$\psi \in F_M(\varphi) \quad \iff \quad \forall q' \colon \mathbf{Q}', \exists n \colon \mathbb{N}, M(n, \varphi, q') = \operatorname{Some} \psi(q').$$

(`operator` in the library with notation `\F_( _ )`). In the same way as was done for the unrelativized case, one may force single-valuedness.

Example 4 (`examples/continuous_search.v`) If $M \colon \mathbb{N} \times \mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \operatorname{opt} \mathbb{N}$ is taken to be the function that returns Some k if k is the smallest number no bigger than n such that $\varphi(k) = 0$ and None if no such k exists, then F_M is the search operator whose domain are the functions that eventually hit zero and whose value is the constant function returning the minimal such number.

As a word of warning, and an additional motivation for the next section, let us briefly look into composition of computable functions and operators. First consider the assignment $\Phi.$ Given functions $N \colon \mathbb{N} \times R \to \operatorname{opt} S$ and $N' \colon \mathbb{N} \times S \to \operatorname{opt} T$ one may define a new function $N' \circ_\Phi N \colon \mathbb{N} \times R \to \operatorname{opt} T$ via

$$N' \circ_\Phi N(\langle n, m \rangle, r) := \begin{cases} N'(n, t) & \text{if } N(m, r) = \operatorname{Some} t \\ \text{None} & \text{otherwise,} \end{cases}$$

where $\langle \cdot, \cdot \rangle$ is the Cantor (or any standard) paring function. This captures the relational composition in the sense that $\Phi_{N' \circ_\Phi N} = \Phi_{N'} \circ_R \Phi_N$ which in turn tightens $\Phi_{N'} \circ \Phi_N$. Under the assumption that N is monotone, one may even simplify the construction and avoid the need to use a pairing function (the construction is included in the CoqRep library).

In the relativized case we are interested in composing operators and in obtaining a tightening of $F_{M'} \circ F_M$ from M' and M. To understand why we cannot simply use an analogue of above construction, fix some $M\colon \mathbb{N} \times \mathcal{B} \times \mathbf{Q}' \to \operatorname{opt} \mathbf{A}'$ and $M'\colon \mathbb{N} \times \mathcal{B}' \times \mathbf{Q}'' \to \operatorname{opt} \mathbf{A}''$. Due to the definition of the F_M assignment, we can recover from M finite approximations to possible functional inputs to M'. However, to get any information about the return value of the composition it is necessary to extend these finite approximations to a total function that can be used as input to M' and without further information it is not clear why the return values of M' on an extension should have anything to do with a return-value of the composition. In particular it is not clear that it can be made independent of how we extend. Of course, since Coq is consistent with functional extensionality and any function in Coq can be evaluated in a finite amount of time, one might tend to believe that if M' can be defined without use of axioms, its value on fixed inputs only relies on a finite number of the return values of its functional input. The additional information that is needed about M' for being able to carry out the composition is exactly what is introduced as a modulus of continuity in the upcoming section.

2.2 Continuity of partial operators between Baire spaces

This section presents an information theoretic development of a notion of continuity of operators between Baire spaces. The Incone library provides proofs that the definitions presented here are equivalent to more traditional notions of continuity, but the discussion of these equivalences is postponed to Section 4 since it requires some background about metric spaces and topology that are not necessary for the presentation in the current section. For the following fix some types $\mathbf{Q}$, $\mathbf{A}$, $\mathbf{Q}'$ and $\mathbf{A}'$ and set $\mathcal{B} := \mathbf{Q} \to \mathbf{A}$ and $\mathcal{B}' := \mathbf{Q}' \to \mathbf{A}'$.

Intuitively continuity means that the return-values of an operator $F\colon \mathcal{B} \to \mathcal{B}'$ interpreted as functional of type $F\colon \mathcal{B} \times \mathbf{Q}' \to \mathbf{A}$ do only depend on finite information about the values of the functional input from $\mathcal{B}$ and thus can be thought of as being represented by a diagram as depicted in Figure 1. Mathematically, continuity can be described as follows: A function $F\colon \mathcal{B} \to \mathcal{B}'$ is **continuous** if for any element φ of $\mathcal{B}$ and any $q'\colon \mathbf{Q}'$ there exists a **certificate**, i.e., a finite list $L\colon \operatorname{seq} \mathbf{Q}$ such that for any ψ that coincides with φ on L it holds that $F(\psi)(q') = F(\varphi)(q')$. Here, two functions are said to **coincide on** a finite list L if $\varphi(q) = \psi(q)$ for any q contained in L. A partial operator $F\colon \subseteq \mathcal{B} \to \mathcal{B}'$ is continuous if for all $\varphi \in \operatorname{dom}(F)$ and $q'\colon \mathbf{Q}'$ there exists a certificate, i.e., a finite list $L \subseteq \mathbf{Q}$ such that the above statement holds for any $\psi \in \operatorname{dom}(F)$.

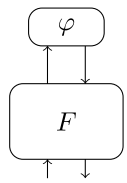

Figure 1: A continuous operator.

Example 5 (`examples/continuous_search.v`) Functions that can be defined in Coq without mentioning any axioms are usually easy to prove continuous. For instance consider the function $F_0\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ defined by $F_0(\varphi)(n) := \varphi(n) + \varphi(0)$. Then obviously, for any inputs φ and n the finite list $L := [n; 0]$ will do. For the operator $F_1(\varphi)(n) := \varphi(\varphi(n))$ the list $L := [\varphi(n); n]$ is appropriate.

The meta-level explanation of the behavior described in this example is that any function definable in Coq without axioms is computable and therefore continuous.

Example 6 (`examples/continuous_search.v`) The same remains true for operators whose specification can be given as F_M for some function $M\colon \mathbb{N} \times \mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \operatorname{opt} \mathbb{N}$ such that M has a Coq-definition that is closed under the global context. For instance for the search operator F from Example 4 the function M can be defined in Coq without axioms and the list $[0; \ldots ; F(\varphi)(0)]$ is a certificate for functional input φ and discrete input n. The search operator does not have a continuous total extension. The fairly vague argument that any Coq function is computable and total, and that any computable function is continuous makes it reasonable to assume that no choice function for this specification can be defined in Coq without relying on axioms.

Indeed, from a meta-level many of the proofs of continuity that can be found in the INCONE library proceed by specifying an axiom-free COQ-function interpreted either through the `F2MF` or through the $F.$ assignment and may thus be understood as proofs of computability whenever the open types are instantiated appropriately (at the very least if all input and output spaces are set to be the natural numbers). All claims of computability in the rest of the paper should be understood in this sense.

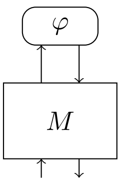

Figure 2: A computable operator

The definition of continuity in the INCONE library follows the mathematical definition given earlier mostly literally. The only difference being that instead of a separate list for each $q' \colon \mathbf{Q}'$ a Skolem-function $\mu \colon \mathbf{Q}' \to \operatorname{seq} \mathbf{Q}$ is used, which switches the order of the corresponding existential and universal quantification. This is equivalent to the above definition whenever an appropriate choice principle is available (`choice_cont`) and avoids assuming any choice principles in the proof that the composition of continuous operators is continuous.

Partiality is treated by using multifunctions and the statement of continuity of a multifunction is chosen in such a way that continuity implies the function to be single-valued (`cont_sing`). The notion of a certificate is made sense of for general multivalued functions (we mention this because it is used below). Continuity for multifunctions on Baire space should not be confused with the notion of continuous realizability that is introduced in Section 3 as it is different in character. Still it works well with the composition of multivalued functions, as that reproduces the usual composition of partial functions.

Theorem 7 (`cont_comp`) *Let $F \colon \subseteq \mathcal{B} \to \mathcal{B}'$ and $G \colon \subseteq \mathcal{B}' \to \mathcal{B}''$ be continuous partial operators. The operator $F \circ G \colon \subseteq \mathcal{B} \to \mathcal{B}''$ is continuous.*

The idea behind the proof is that the certificate functions μ and ν whose existence is guaranteed by the continuity of F and G can be interpreted as multivalued functions and composed relationally to obtain a certificate function for the composition of the operators. Furthermore, the needed relational composition can be realized constructively on the level of combining lists.

As we compare different notions of continuity in the later chapters, let us briefly discuss sequential continuity on Baire spaces. An element φ of a Baire space is said to be the **limit** of a sequence (φ_n) in $\mathcal{B}$ if for each fixed argument $q \colon \mathbf{Q}$ the sequence $(\varphi_n(q))$ is eventually constantly $\varphi(q)$. Formally

$$\lim_{\mathcal{B}}(\varphi_n) = \varphi \quad \iff \quad \forall q, \exists N, \forall n \colon \mathbb{N}, N \leq n \implies \varphi_n(q) = \varphi(q).$$

If a sequence in Baire space has a limit, this limit is uniquely determined (`lim_sing`) and thus the above defines a partial function $\lim_{\mathcal{B}} \colon \subseteq (\mathbb{N} \to \mathcal{B}) \to \mathcal{B}$.

A partial operator $F \colon \subseteq \mathcal{B} \to \mathcal{B}'$ is called **sequentially continuous** if for any $\varphi \in \operatorname{dom}(F)$ and any sequence (φ_n) from the domain of the operator such that $\lim_{\mathcal{B}}(\varphi_n) = \varphi$ it also holds that $\lim_{\mathcal{B}'}(F(\varphi_n)) = F(\varphi)$. It is well known that the topological structure of Baire space is such that sequential continuity is equivalent to continuity and the INCONE library includes a formal proof of this. However, this is a classical fact and a constructive proof of sequential continuity provides strictly less information than a proof of continuity, thus the library separates the equivalence in into two implications.

Theorem 8 (`cont_scnt` **and** `scnt_cont`) *A partial operator between Baire spaces is continuous if and only if it is sequentially continuous.*

Section 4.1 equips Baire space with the structure of a metric space and recovers the notions of a limit and continuity discussed here from the metric notions. Thus, the formal proofs that continuity and sequential continuity are identical to what is described in Section 4.2 imply the above theorem. However, metric spaces use real numbers, which leads to the axioms of the real numbers appearing in the assumptions printed when inspecting the proofs. This is even though the proofs do not use these axioms in an essential way. Thus, the above statement is given a separate proof in the INCONE library.

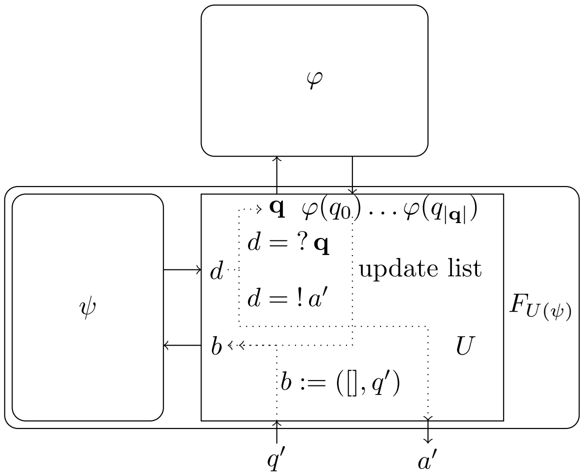

Figure 3: The universal used in the library.

2.3 Construction of a universal and proofs of some of its properties

Let $\mathcal{B}$ and $\mathcal{B}'$ be Baire spaces in the sense of the previous sections. A **continuous universal**, or just **universal**, is a pair of a Baire space $\mathcal{B}''$ and an operation $F_{M(\cdot)} \colon \mathcal{B}'' \to \mathcal{B} \rightrightarrows \mathcal{B}'$ such that for each continuous $F \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ there exists an element $\psi \in \mathcal{B}''$ such that $F_{M(\psi)}$ tightens F. That is: a universal provides a way to code any continuous operator between Baire spaces by an element of another Baire space. To motivate the name, note that a universal Turing machine fulfills a modification of the above, where all the Baire spaces are replaced by the set of finite binary strings and the word "continuous" by "computable". While the construction of a universal Turing machine takes some effort, continuous universals can be chosen very simple: Well-known continuous universals are the Kleene-Kreisel associate construction [Kle59, Kre59] and Weihrauch's η operator [Wei00]. The former of these is conceptually more well adapted to our setting and there are several excellent sources to read up about its background [LN15, EX16]. Our first implementation of a universal followed the Kleene-Kreisel construction very directly and is still used in the CoqRep library.

The universal that is currently used in the Incone library is a slight modification of the original construction for the sake of efficiency. Namely it uses lists as a simple way of avoiding the need to take too many loops through the universal that may lead to reevaluation and are generally a waste of time. An exact specification is the following (compare Figure 3): for fixed $\mathcal{B}$ and $\mathcal{B}'$ set $\mathbf{Q}'' := \operatorname{seq} \mathbf{A} \times \mathbf{Q}'$ and $\mathbf{A}'' := \operatorname{seq} \mathbf{Q} + \mathbf{A}'$, that is use $\mathcal{B}'' := \operatorname{seq} \mathbf{A} \times \mathbf{Q}' \to \operatorname{seq} \mathbf{Q} + \mathbf{A}'$. That $\mathcal{B}''$ is a Baire space, i.e., that $\mathbf{Q}''$ and $\mathbf{A}''$ are countable and inhabited, follows directly from $\mathcal{B}$ and $\mathcal{B}'$ being Baire spaces. Assign to a function ψ from the Baire space $\mathcal{B}'' = \operatorname{seq} \mathbf{A} \times \mathbf{Q}' \to \operatorname{seq} \mathbf{Q} + \mathbf{A}'$ the multifunction $F_{U(\psi)} \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ defined as follows: $\varphi' \in F_{U(\psi)}(\varphi)$ if and only if for any $q' \in \mathbf{Q}'$ there exists a finite sequence of lists $(L_i)_{i \in \{1,\ldots,N\}} \subseteq \operatorname{seq} \mathbf{A}$ such that for $i < k$ it holds that $\psi(L_i, q') = ?\,\mathbf{q}$ for some $\mathbf{q} \in \operatorname{seq} \mathbf{Q}$ (where ? denotes the left inclusion in the sum) and $L_{i+1} = L_i \varphi(q_1) \ldots \varphi(q_{|\mathbf{q}|})$ and $\psi(L_N, q') = !\,\varphi'(q')$ (where ! denotes the right inclusion of the sum).

The notation we used for continuous universals heavily implies that the universal can be specified by means of a function $U \colon \mathcal{B}'' \to \mathbb{N} \times \mathcal{B} \times \mathbf{Q}' \to \operatorname{opt} \mathbf{A}'$ where the universal as described above can be recovered using the interpretation described in the first subsection via $\psi \mapsto F_{U(\psi)}$. Indeed, such a function `U` is specified in the library and its Coq-definition is closed under the global context. We refrain from writing out the exact definition of U here and point the interested reader to the Incone library, where the definition can be printed and a formal proof that it fulfills the above specification can be found (`FU_spec`). Furthermore, `U`(ψ) is always monotone in the sense of the previous section (`U_mon`) and in particular $F_{U(\psi)}$ is always single-valued (`FU_sing`).

Define the **multivalued modulus of continuity** $L_F \colon \mathcal{B} \rightrightarrows (\mathbf{Q}' \to \operatorname{seq} \mathbf{Q})$ of an operator $F \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ (`continuity_modulus` in the library) by

$$\mu \in L_F(\varphi) \quad \iff \quad \forall q' \colon \mathbf{Q}', \mu(q') \text{ is a certificate for } \varphi \text{ and } q'.$$

The definition of L_F makes sense for any multifunction and $F \colon \mathcal{B} \to \mathcal{B}'$ is continuous, if and only if the domain of L_F is a super-set of the domain of F (`cont_spec`). If F is continuous, then we call any partial choice function of L_F a **modulus of continuity** of F. Note that the multivalued modulus of continuity of a modulus of continuity has the same type as L_F. Thus it makes sense to call a modulus of continuity of F self-modulating if it is its own modulus of continuity.

Whenever ψ is such that $F_{U(\psi)}$ tightens F, a self-modulating modulus of F can readily be obtained by tracking the queries in the evaluation of the universal. The same can be done for the values that the universal calls the function ψ on and one defines such functions U_Q and U_S (`queriesM` and `shapesM` in the library with definitions that are closed under the global context) such that $F_{U_Q(\psi)}$ and $F_{U_S(\psi)}$ are the corresponding operators.

Theorem 9 (`FqM_mod_FU`, `FqM_mod_FqM` and `FqM_mod_FsM`) *For any fixed* $\psi \colon \operatorname{seq} \mathbf{A} \times \mathbf{Q}' \to \operatorname{seq} \mathbf{Q} + \mathbf{A}'$ *the operator* $F_{U_Q(\psi)}$ *is a modulus of continuity of* $F_{U(\psi)}$*, of itself and of* $F_{U_S(\psi)}$.

The universal is used in the library to construct function spaces, that is exponentials, in the category of represented spaces. It should be reasonable that the above is essential for proving the evaluation procedure on the constructed space of functions to be a continuous operation. It also implies that for any ψ the operator $F_{U(\psi)}$ is continuous (`FU_cont`). While the functions U_Q and U_S are of theoretical importance, for the practical purpose of inspecting the behavior of an associate the library provides additional functions `gather_queries` and `gather_shapes` that return the queries done up to the n-th loop of the universal (see `examples/KleeneKreisel.v`).

The library also provides a full proof that U specifies a universal:

Theorem 10 (`U_universal`) *For any countable and non-empty types* $\mathbf{Q}$*,* $\mathbf{A}$*,* $\mathbf{Q}'$ *and* $\mathbf{A}'$ *and for any continuous operator* $F \colon (\mathbf{Q} \to \mathbf{A}) \rightrightarrows (\mathbf{Q}' \to \mathbf{A}')$ *there exists some* $\psi \colon \operatorname{seq} \mathbf{A} \times \mathbf{Q}' \to \operatorname{seq} \mathbf{Q} + \mathbf{A}'$ *such that* $F_{U(\psi)}$ *tightens* F.

While we do not repeat the proof given in the library here, let us sketch the most important parts and point out some interesting details. Fix some enumeration of $\mathbf{Q}$, let μ be a function that chooses through the multivalued modulus of continuity L_F of F and let f be a function that chooses through F. For any fixed enumeration of $\mathbf{Q}$, one can attempt to define an associate ψ (`psiF` in the library) for F as follows: On input (L, q') interpret the list L as a partial function by assuming that its elements are the return values on the first $|L|$ elements mentioned in the enumeration of $\mathbf{Q}$. Extend this function to a total function φ_L that is from the domain of F if this is possible. Check whether each element of $\mu(\varphi_L, q')$ is contained in the $|L|$ first elements mentioned in the enumeration of $\mathbf{Q}$. If this is so return $!\, f(\varphi_L)(q')$, otherwise ask for the $(|L|+1)$-st element mentioned in the enumeration.

Without further assumptions about μ the function ψ defined above might fail to be an associate of F as the extensions that μ is called on can change in each iteration and μ may return a properly bigger list each time, such that the run of $U(\psi)$ diverges. Since it is always possible to extract a self-modulating modulus from an associate it may not be surprising that what is needed to complete the proof is indeed to construct a self-modulating modulus. In the library the existence of such a modulus is proven by picking μ to be minimal with respect to subset inclusion under the additional condition that it can only return initial segments with respect to the enumeration of $\mathbf{Q}$.

Lemma 11 (`mod_minmod`) *A minimal modulus is always self-modulating.*

The reason for the indefinite article in this lemma is that it is well know, that the existence of a minimal modulus is not constructively provable [TvD88], and indeed our Coq-proof of its existence is classical.

Lemma 12 (`exists_minmod`) *Any continuous operator has a minimal modulus of continuity.*

The INCONE library makes some efforts to decompose the proof of Theorem 10 into components, such that the main parts are constructive and can be used to extract an associate from additional information that has to be provided explicitly. Currently, this additional information is a choice function for the operator, and a self-modulating modulus of the operator together with a choice function for it and a function that extends a partially defined input to a total input that is from the domain of the operator if this is possible. It is reasonable to assume that the amount of information can be minimized quite a bit more, as the above can be interpreted as a partial recovery of one implication of the well-known fact that there exists a computable ψ if and only if F is computable.

The other implication, that if ψ is computable then so is $F_{U(\psi)}$, can indeed be fully recovered with respect to our meta-level notions of computability from Section 2.1: Whenever ψ can be defined in COQ without axioms, the term $U(\psi)$ is also axiom-free and evidence that $F_{U(\psi)}$ is computable. The assumption that ψ is expressible in axiom free COQ is a priori stronger than computability, but if an axiom free $N\colon \mathbb{N} \times \operatorname{seq} \mathbf{A} \times \mathbf{Q}' \to \operatorname{opt}(\operatorname{seq} \mathbf{Q} + \mathbf{A}')$ is given such that $\Phi_N = \psi$, a COQ-definition of a function ψ' such that $F_{U(\psi)} = F_{U(\psi')}$ can be recovered: Let $q \in \mathbf{Q}$ be the witnesses that $\mathbf{Q}$ is inhabited and let ψ' map a pair (L, q') to the phantom query $?[q]$ if $N(|L|, L', q') = \text{None}$, where L' is a modified list L where the values corresponding to previous phantom queries are removed, and to r if $N(|L|, L', q') = \text{Some}\, r$.

Even though inconsequential for the rest of the paper, we feel that it is worth mentioning that the INCONE library defines a function D that exchanges the arguments of the universal.

Theorem 13 (`D_spec`) *For all φ and ψ it holds that $F_{U(\psi)}(\varphi) = F_{U(D\varphi)}(\psi)$.*

Here, the types have been purposefully omitted, details can be found in the library. The important point is that the function D is defined axiom-freely. Note that, while $U(\psi)$ has the more complicated type and is interpreted using as $F_{U(\psi)}$, the operator D can be realized by a COQ-function directly, i.e., using the `F2MF` correspondence. Indeed, the definition of D can be expressed by minimal means, and the operator should be considered primitive recursive. The above theorem is interesting because it is related to the Cartesian closure of the category of represented spaces (see Section 3 for details on the category). However, it falls slightly short in strength as it only considers a special case in which it is not necessary to talk about tupling of elements of Baire spaces.

Corollary 14 (`FsM_mod_FU`, `FsM_mod_FsM` and `FsM_mod_FqM`) *Theorem 9 remains true if U_Q and U_S are exchanged and ψ and φ are exchanged.*

The above two cases where it was possible to eliminate the more complicated meta-level notion of computability in favor of `F2MF` are not coincidental. They can be understood as a consequences of the fact that the function space construction from computable analysis is such that it equips any space of functions with a very specific kind of representation [KP14, Ste17, NS17]. In particular the representation of a space of functions is always pre-complete [KW85], but in this context it seems that pre-completeness is not exactly the appropriate notion.

3 Represented spaces and continuous realizability

Recall that a realizability relation is a relation between data and abstract objects that assigns to each abstract object a nonempty set of data points that implement it. Computable analysis picks Baire space as the set of data and interprets a realizability relation as a specification of a function from data to abstract objects. Thus, computable analysis reasons about co-total multifunctions $\delta\colon \mathcal{B} \rightrightarrows X$ called multi-representations. Most of computable analysis is only concerned with the special case where these multifunctions are single-valued. A **representation** δ of a space X is a partial surjective mapping $\delta\colon \subseteq \mathcal{B} \to X$. The φ from $\mathcal{B}$ such that

$\delta(\varphi) = x$ are called the δ-names, or just names, of x. A pair $\mathbf{X} = (X, \delta_{\mathbf{X}})$ of a set and a representation of that set is called a **represented space**.

The definition in the INCONE library replaces the Baire space $\mathbb{N}^{\mathbb{N}}$ from the definition used in computable analysis with some space $\mathcal{B} = \mathbf{Q} \to \mathbf{A}$, where $\mathbf{Q}$ and $\mathbf{A}$ should be countable inhabited types, i.e., with a Baire space in the sense in which the phrase was used throughout Section 2. Thus, a represented space $\mathbf{X}$ is defined as a record containing a type X (with a coercion from $\mathbf{X}$ to X that explains the special notation without index) together with types $\mathbf{Q}_{\mathbf{X}}$ and $\mathbf{A}_{\mathbf{X}}$ and proofs that these are countable and inhabited and additionally a multivalued function $\delta_{\mathbf{X}} \colon (\mathbf{Q}_{\mathbf{X}} \to \mathbf{A}_{\mathbf{X}}) \rightrightarrows X$ and proofs that it is single-valued and co-total (which is equivalent to being surjective by Lemma 2). We use the notation $\mathcal{B}_{\mathbf{X}} := \mathbf{Q}_{\mathbf{X}} \to \mathbf{A}_{\mathbf{X}}$ (`names` in the library).

As an example let us equip the real numbers with a representation that is used for motivation and as benchmark throughout this section.

Example 15 (`examples/Q_reals.v`) Choose $\mathbf{Q}_{\mathbb{R}} = \mathbf{A}_{\mathbb{R}} := \mathbb{Q}$, i.e., $\mathcal{B}_{\mathbb{R}} := \mathbb{Q} \to \mathbb{Q}$. It is straight forward to prove that the type for rational numbers provided by COQ's standard library is countable and inhabited. Furthermore, the multifunction $\delta_{\mathbb{R}} \colon \mathcal{B}_{\mathbb{R}} \rightrightarrows \mathbb{R}$ (`rep_RQ` in the library) given by:

$$\delta_{\mathbb{R}}(\varphi) = x \qquad \iff \qquad \forall \varepsilon \in \mathbb{Q}, 0 < \varepsilon \implies |x - \varphi(\varepsilon)| < \varepsilon$$

is a representation. Indeed, using the axiomatization of the real numbers provided by COQ's standard library $\delta_{\mathbb{R}}$ can be proven single-valued and surjective (`rep_RQ_sing` and `rep_RQ_sur`) and we refer to the represented space $(\mathbb{R}, \delta_{\mathbb{R}})$ (`RQ` in the library) simply by $\mathbb{R}$.

The topological and computability structure of Baire space can be pushed forward through a representation: A partial operator on Baire space is a **realizer** of a function $f \colon \mathbf{X} \to \mathbf{Y}$ between represented spaces if it assigns to each name of x a name of $f(x)$ (compare Figure 4). A function between represented spaces is **continuous** if it has a continuous realizer and **computable** if it has a computable realizer. In computable analysis it is well-known that the represented spaces form a Cartesian closed category both if the continuous functions are used as morphisms, as well as if the computable functions are used.

$$\begin{tikzcd} X \arrow[r, "f"] & Y \\ \mathcal{B} \arrow[u, "\delta_{\mathbf{X}}"] \arrow[r, "F"'] & \mathcal{B}' \arrow[u, "\delta_{\mathbf{Y}}"'] \end{tikzcd}$$

Figure 4: $F \colon \subseteq \mathcal{B} \to \mathcal{B}'$ is a realizer of $f \colon \mathbf{X} \to \mathbf{Y}$

With little effort, the definition of being a realizer can be made sense of if both operators on Baire space and functions between represented spaces are multivalued and can even be generalized to multi-representations. For the full definitions we point the interested reader to the RLZRS library, in the case where the representations are single-valued, the appropriate definitions can also be recovered from the following characterization:

Lemma 16 (`rlzr_spec`) *let* $\mathbf{X}$ *and* $\mathbf{Y}$ *be represented spaces,* $F \colon \mathcal{B} \rightrightarrows \mathcal{B}'$ *realizes* $f \colon \mathbf{X} \rightrightarrows \mathbf{Y}$ *if and only if* $\delta_{\mathbf{Y}} \circ F$ *tightens* $f \circ \delta_{\mathbf{X}}$.

The above lemma can be used backwards to express the notion of tightening as a special case of realizability by using the identity function on the spaces as realizability relation (`id_rlzr_tight`). For concrete proofs other lemmas that simplify the realizer relation in the case where additional information about the realizing or the realized multifunction is available are often more useful (`sing_rlzr_F2MF`, `F2MF_rlzr_F2MF`, etc.).

While we are mostly interested in continuous, and therefore single-valued realizers, the case where f is multivalued is of interest to us as it is needed for the concrete example of closed choice on the natural numbers that we discuss in Section 4.3. We call a multifunction between represented spaces **continuously realizable** if there exists a continuous realizer in this sense (`hcr` in the library with notation `_ \has_continuous_realizer`). An standard example where the use of a multifunction instead of a function can be used to recover continuity is that while the characteristic sign function is discontinuous, it can be approximated by the family of continuous ε-sign multifunctions whose value is allowed to be either -1 or 1 whenever $|x|$ is smaller than ε or a similar ε-equality test to account for the undecidability of equality on the real numbers. That continuity and continuous realizability is preserved under composition follows from content of the Rlzrs library together with the fact that continuity of operators on Baire-space is preserved under composition from Theorem 7.

Lemma 17 (`comp_cont` and `comp_hcr`) *The composition of continuous functions is continuous and the composition of continuously realizable multifunctions is continuously realizable.*

The notion of continuous realizability of a multifunction between Baire-spaces equipped with the identity representations does only coincide with continuity as introduced in Section 2.2 if restricted to single-valued functions. While a partial function is continuously realizable if and only if it is continuous, there are many multivalued functions that are continuously realizable but not continuous. This is because continuity implies single-valuedness and continuous realizability, to the contrary, is stable under increasing the set of eligible return values. The ε-sign and ε-equality test from above can be used to see that a continuously realizable multifunction need not always have a continuous choice function for represented spaces that are not equipped with the identity representation.

3.1 Finite products, sums, lists and basic examples

Now that we can talk about continuity and computability on the real numbers, a reasonable next step is to attempt to prove addition and multiplication computable. Both of these functions are of type $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and to make sense of continuity of functions of these types we need to specify how $\mathbb{R} \times \mathbb{R}$ should be made a represented space. The Incone library automatically generates such a represented space $\mathbf{X} \times \mathbf{Y}$ from arbitrary represented spaces $\mathbf{X}$ and $\mathbf{Y}$ by using the query type $\mathbf{Q}_{\mathbf{X}\times\mathbf{Y}} := \mathbf{Q}_{\mathbf{X}} + \mathbf{Q}_{\mathbf{Y}}$, the answer type $\mathbf{A}_{\mathbf{X}\times\mathbf{Y}} := \mathbf{A}_{\mathbf{X}} \times \mathbf{A}_{\mathbf{Y}}$ and the representation $\delta_{\mathbf{X}\times\mathbf{Y}}$ defined by

$$\delta_{\mathbf{X}\times\mathbf{Y}}(\psi) = (x, y) \quad \iff \quad \delta_{\mathbf{X}}(\mathrm{fst} \circ \psi \circ \mathrm{inl}) = x \wedge \delta_{\mathbf{Y}}(\mathrm{snd} \circ \psi \circ \mathrm{inr}) = y.$$

This can be decoded as follows: A name of the pair (x, y) should be a pair (φ, φ') of a name for x and a name for y. Since the set of pairs $\mathcal{B}_{\mathbf{X}} \times \mathcal{B}_{\mathbf{Y}}$ does not have the type that we required a Baire space to have, we embed it into the Baire space $\mathcal{B}_{\mathbf{X}\times\mathbf{Y}} := \mathbf{Q}_{\mathbf{X}} + \mathbf{Q}_{\mathbf{Y}} \to \mathbf{A}_{\mathbf{X}} \times \mathbf{A}_{\mathbf{Y}}$. There are several possible choices for $\mathcal{B}_{\mathbf{X}\times\mathbf{Y}}$ and the one we picked is not the minimal one. However, for our pick the projection functions on the Baire space can particularly naturally be expressed by the natural operations on the input and output spaces, namely by the first one by $\psi \mapsto \mathrm{fst} \circ \psi \circ \mathrm{inl}$ and the other one analogously.

Proposition 18 (`prod_rep_sing`, `prod_rep_sur` and `prod_uprp_cont`) *For any represented spaces $\mathbf{X}$ and $\mathbf{Y}$ the space $(X \times Y, \delta_{\mathbf{X}\times\mathbf{Y}})$ is a represented space and it is the product of $\mathbf{X}$ and $\mathbf{Y}$ in the category of represented spaces.*

That the projections are computable is a sub-task of proving the universal property (`fst_cont` and `snd_cont`). Some other basic functions on product spaces are proven computable in the Incone library, most notably it provides the possibility to glue continuous functions on two factors to a continuous function between products (`fprd_cont`).

Example 19 (`examples/Q_reals.v`) Addition and multiplication of real numbers is computable (`Rplus_cont` and `Rmult_cont`). As described in more detail in Section 2.1 this should be taken to mean that the operations are continuous and the realizers can explicitly be specified as Coq-functions whose definitions contain no axioms. Indeed, the realizers are defined not through the more complicated $F.$ assignment but more directly using the `F2MF` correspondence. Furthermore, their definition only uses very simple tools and the operations should therefore even be considered primitive recursive.

As another basic example of a represented space that is needed below let I be any countable and inhabited type. Set $\mathbf{Q_I} := \{\star\}$ and $\mathbf{A_I} := I$. Then the function $\delta_\mathbf{I}(\varphi) := \varphi(\star)$ makes $(I, \delta_\mathbf{I})$ a represented space that is discrete in the following sense:

Lemma 20 (`cs_id_dscrt`) *For any countable, inhabited type I the represented space $\mathbf{I}$ described above is discrete in the sense that any function that has $\mathbf{I}$ as its domain is continuous. Moreover, any multivalued function with $\mathbf{I}$ as input space is continuously realizable.*

In particular, the natural numbers can be assigned a discrete represented space that we denote by $\mathbb{N}$ in the following.

The Incone library proves that the represented space $\mathbf{1}$ constructed from the unit type as above is a terminal object in the category of represented spaces. Furthermore it provides an option type construction and proves that the option type over $\mathbf{X}$ is always isomorphic to $\mathbf{X} + \mathbf{1}$. The constructions lack good use cases and we omit the details. For the same reasons we omit the construction of the sum space $\mathbf{X} + \mathbf{Y}$ from represented spaces $\mathbf{X}$ and $\mathbf{Y}$ which is for the most part analogous to the product. Instead we describe how one could construct a represented space $\operatorname{seq}\mathbf{X}$ of finite lists of elements from a given represented space $\mathbf{X}$. Set $\mathbf{Q}_{\operatorname{seq}\mathbf{X}} := \mathbf{Q_X}$, $\mathbf{A}_{\operatorname{seq}\mathbf{X}} := \operatorname{seq}\mathbf{A_X}$ and consider the representation defined by

$$\delta_{\operatorname{seq}\mathbf{X}}(\varphi) = L \quad \iff \quad \forall i \in \{1, \ldots, |L|\}, \delta_\mathbf{X}(q \mapsto \varphi(q)_i) = L_i.$$

This representations is not defined in the Incone library and has not been proven correct. A similar representation is defined in the CoqRep library and while it is equivalent to the one given here, its definition is more complicated for historical reasons.

The reason why we still go into details about lists is that the CoqRep library proves custom-made and not fully general induction principles for $\mathbb{N}$ and for lists. It also define a represented space $\mathbb{R}[X]$ of polynomials and uses the induction principles to prove polynomial evaluation computable as function of type $\mathbb{R}[X] \times \mathbb{R} \to \mathbb{R}$. The expected performance of the resulting algorithm is very poor: While it does use the Horner scheme, which at least avoids that approximations to the same real number are computed a quadratic number of times, it still computes a linear number of approximations to the same number. In this case computing a single approximation with the highest accuracy requested during the computation would have been sufficient. For spaces different from the real numbers it might be necessary to ask many different questions. Clearly, this is something that is difficult to figure out for a general purpose induction principle that is not provided with additional information about the representation. In many cases it may be reasonable to handcraft the algorithms. For the real numbers there exist general purpose tools to better the performance of algorithms based on arithmetic operations [Mül01].

Thus, even-though it is possible to prove induction principles at this point, it is questionable whether one wants to make such available as general purpose tools for generating algorithms. It may be a good idea to first understand in which cases they are prone to introduce inefficiencies and how this can be avoided [NS17].

3.2 Infinite products, limits and point-wise operations

Let I be a countable inhabited type and let $\mathbf{X}$ be a represented space. Define a represented space $\prod_I \mathbf{X}$ whose underlying set are the functions of type $I \to X$ by setting $\mathbf{Q}_{\prod_I \mathbf{X}} := I \times \mathbf{Q_X}$, $\mathbf{A}_{\prod_I} := \mathbf{A_X}$ and

$$(x_i) \in \delta_{\prod_I \mathbf{X}}(\varphi) \quad \iff \quad \forall i \colon I, x_i \in \delta_\mathbf{X}(q \mapsto \varphi(i, q)),$$

where (x_i) is short for the function $i \mapsto x_i$.

Proposition 21 (`rep_Iprod_sing`, `rep_Iprod_sur` **and** `cprd_uprp_cont`) *For countable inhabited I the space $\prod_I \mathbf{X} := (I \to X, \delta_{\prod_I \mathbf{X}})$ is a represented space. The space $\mathbf{X}^\omega := \prod_\mathbb{N} \mathbf{X}$ is the countably infinite product in the category of represented spaces and continuous functions.*

The use of the symbol ω instead of $\mathbb{N}$ is to differentiate the space $\mathbf{X}^\omega$ of sequences (notation `_ \^w` in the library) from the function space discussed in the next section. The proof of single-valuedness assumes functional extensionality and the proof of surjectivity needs a choice principle over I. Since $I = \mathbb{N}$ is by far the most common use-case and I is assumed to be countable anyway, this will usually boil down to the axiom of countable choice. The proof of the universal property relies on stronger choice principles, classical reasoning and proof irrelevance. Since the category of represented spaces with computable functions fails to have countably infinite products, the universal property should not be provable without axioms. In how far our use of axioms can be optimized in this case is difficult to tell at the this point in time since the current proof uses a part of the library that has not yet been optimized in terms of axiom use. Since it is more a sanity result than something that may actually be of use, optimizations here are not our highest priority.

An example of a partial function whose natural domain is a subset of the space of sequences is the limit operator. Consider the multivalued function $\lim_{\mathbf{X}} : \mathbf{X}^\omega \rightrightarrows \mathbf{X}$ where $x \in \lim_{\mathbf{X}}(x_n)$ if and only if there is a convergent sequence of names $(\varphi_n) \subseteq \mathcal{B}_{\mathbf{X}}$ and some φ such that φ is a name of x, each φ_n is a name for x_n and the sequence (φ_n) converges to φ in $\mathcal{B}_{\mathbf{X}}$, i.e., $\lim_{\mathcal{B}_{\mathbf{X}}}(\varphi_n) = \varphi$ where the limit in $\mathcal{B}_{\mathbf{X}}$ is taken point-wise as explained in Section 2.2. While the limit operator on Baire space is single-valued, this need not be true for the limit operator on a general represented space, as can be seen at the example of Sierpiński space that is discussed in Section 4.3. In most spaces that are relevant for numerical analysis, the limit operator is single-valued.

Example 22 (`examples/Q_reals.v`) The limit operator $\lim_{\mathbb{R}}$ where $\mathbb{R}$ is represented as in Example 19 is discontinuous (`lim_not_cont` and Section 4.2). Its restriction to those sequences (x_n) that are efficiently Cauchy in the sense that $|x_n - x_m| \leq 2^{-n} + 2^{-m}$ is computable (`lim_eff_hcr` and Section 4.2).

A function $f : \mathbf{X} \to \mathbf{Y}$ between represented spaces is called sequentially continuous if it preserves this notion of a limit, i.e., if $\lim_{\mathbf{X}} x_n = x$ implies that $\lim_{\mathbf{Y}} f(x_n) = f(x)$. In computable analysis it is well known that in general sequential continuity is a weaker notion than continuous realizability in the sense that for some represented spaces $\mathbf{X}$ and $\mathbf{Y}$ there exist sequentially continuous functions that are not continuously realizable. This difference can be eliminated by assuming admissibility of the underlying representations [Sch02a]. We do not go into detail about the notion of admissibility here, but the relation between sequential continuity and continuous realizability is discussed in more detail in Section 4.2.

The library proves some further lemmas about infinite products that might be useful in applications and should thus not go unmentioned. Two represented spaces $\mathbf{X}$ and $\mathbf{Y}$ are **isomorphic**, in symbols $\mathbf{X} \simeq \mathbf{Y}$ (notation `_ ~=~ _` in the library) if there exists a continuous bijection with continuous inverse. The spaces are computably isomorphic if there exists a computable bijection with computable inverse.

Lemma 23 (`cprd_prd`) *For any represented spaces $\mathbf{X}$ and $\mathbf{Y}$ and for any countable inhabited I it holds that $\prod_I(\mathbf{X} \times \mathbf{Y}) \simeq \prod_I \mathbf{X} \times \prod_I \mathbf{Y}$, the spaces are even computably isomorphic.*

The realizers are defined using very limited means and interpreted using the `F2MF` assignment, thus they should be considered primitive recursive.

Any function $f : \mathbf{X} \to \mathbf{Y}$ can be extended to a function $f_I : \prod_I \mathbf{X} \to \prod_I \mathbf{Y}$ defined by $f_I((x_i)) := (f(x_i))$ (`ptw` in the library).

Lemma 24 (`ptw_cont`) *The function f_I is continuous whenever f is a continuous function.*

This lemma has a multivalued variant (`ptw_hcr`). Additionally it can be extended to talk about computability on a meta-level. If the realizer of f can be expressed as a Coq-function through the `F2MF` interpretation, then so can f_I. The details of this are carried out in the CoqRep library but were lost in the transition to the Incone library due to the design-decision not to talk about computability internally. The same should hold true if a realizer of f can be expressed via a Coq function through the $F.$ interpretation, although we never carried out the details.

Another common situation is that an operation $*\colon \mathbf{X}\times\mathbf{Y}\to\mathbf{Z}$ is used to construct an operation $*_I\colon \prod_I \mathbf{X}\times\prod_I\mathbf{Y}\to\prod_I\mathbf{Z}$ via $(x_i)*_I(y_i) := (x_i * y_i)$ (`ptw_op` in the library) and a proof that this extension also preserves continuity can be directly obtained from the previous two lemmas.

Corollary 25 (`cptw_op_cont`) *The operation $*_I$ is continuous whenever $*$ is continuous.*

Of course, the functions `ptw` and `ptw_op` in the library take proofs that I is countable, inhabited and of an appropriate choice principle as additional arguments that where suppressed in the above. For instance, point-wise addition and multiplication are continuous operations on $\mathbb{R}^\omega$.

3.3 Function spaces and their connection to infinite products

Let $\mathbf{X}$ and $\mathbf{Y}$ be represented spaces and denote by $\mathbf{Y}^\mathbf{X}$ the collection of all continuously realizable functions from $\mathbf{X}$ to $\mathbf{Y}$. Recall that in Section 2.3 we gave an explicit description U of a continuous universal. The construction of the universal suggests the choice $\mathbf{Q}_{\mathbf{Y}^\mathbf{X}} := \operatorname{seq}\mathbf{A}_\mathbf{X}\times\mathbf{Q}_\mathbf{Y}$ and $\mathbf{A}_{\mathbf{Y}^\mathbf{X}} := \operatorname{seq}\mathbf{Q}_\mathbf{X}+\mathbf{A}_\mathbf{Y}$ and to consider the multivalued function $\delta_{\mathbf{Y}^\mathbf{X}}\colon \mathcal{B}_{\mathbf{Y}^\mathbf{X}} \rightrightarrows \mathbf{Y}^\mathbf{X}$ defined by

$$f\in\delta_{\mathbf{Y}^\mathbf{X}}(\psi) \quad\iff\quad F_{U(\psi)} \text{ realizes } f.$$

From the formal proof that U describes a universal (i.e. Theorem 10) it follows that this multivalued function is co-total. Since functions (as opposed to partial or multifunctions) are uniquely determined by each of their realizers, $\delta_{\mathbf{Y}^\mathbf{X}}$ is also single-valued and thus a representation.

Proposition 26 (`fun_rep_sing` **and** `fun_rep_sur`) *For any represented spaces $\mathbf{X}$ and $\mathbf{Y}$ the space $(\mathbf{Y}^\mathbf{X},\delta_{\mathbf{Y}^\mathbf{X}})$ described above is a represented space.*

The proof of single-valuedness assumes proof irrelevance and functional extensionality. In the library the represented space $\mathbf{Y}^\mathbf{X}$ is denoted by `cs_fun` with the notation `_ c-> _`. It should be pointed out that the definition in the library differs slightly from what was presented here in that the underlying set is taken to be the functions to be the co-domain of the function representation. This way, the proof of surjectivity is axiom free and the number of axioms assumed automatically whenever function spaces are mentioned is kept low. That the set underlying the function space are exactly the continuous functions is proven retroactively (`ass_cont`).

The following can be derived from the continuity properties of the universal from Theorem 9 and the product construction from Section 3.1.

Theorem 27 (`eval_cont`) *Evaluation as operation $\mathbf{Y}^\mathbf{X}\times\mathbf{X}\to\mathbf{Y}$ is computable.*

The function space construction overlaps in its scope with the infinite product construction: For a countable, inhabited index set I and a represented space $\mathbf{X}$, the set underlying the space $\prod_I\mathbf{X}$ is the set of functions from I to $\mathbf{X}$. The space $\mathbf{I}$ generated from I as was done at the end of Section 3.1 is discrete by Proposition 20. This means that all functions starting from $\mathbf{I}$ are continuous and that the sets underlying $\prod_I\mathbf{X}$ and $\mathbf{X}^\mathbf{I}$ are identical. Indeed these spaces are well known to be computably isomorphic and the library provides a formal proof of this fact.

Theorem 28 (`sig_iso_fun`) *For any represented space $\mathbf{X}$ and any countable inhabited type I the space $\prod_I\mathbf{X}$ from the last section is computably isomorphic to the function space $\mathbf{X}^\mathbf{I}$, where $\mathbf{I}$ the canonical discrete space as described at the end of Section 3.1.*

A realizer $f\colon \mathcal{B}_{\prod_I \mathbf{X}} \to \mathcal{B}_{\mathbf{X}^\mathrm{I}}$ that translates a name of a sequence to a name of the corresponding function can directly be specified via

$$f(\varphi)(L, q') := \begin{cases} \star & \text{if } L = [] \\ \varphi(i, q') & \text{if } L = \mathrm{cons}(i, L'). \end{cases}$$

Again, this realizer should be considered primitive recursive. Furthermore, it should be noted that it relies on the implementation of the universal which may be attributed to the fact that the above theorem need not be true in an arbitrary Cartesian closed category. The construction of a continuous function from the space of functions proceeds by using a variation of the realizer of evaluation. On one hand this means that it is independent of the implementation of the universal. On the other it means that the universal has to be executed and thus it is an instance where a realizer uses the more complicated $F.$ assignment. An axiom-free definition of a realizer using the `F2MF` assignment is likely to be impossible for reasons similar to those given for the continuous search operator in Example 4. This is also related to the fact that the construction of the reals from Dedekind cuts and Cauchy sequences are not fully equivalent in a constructive setting [LR08].

4 Metric spaces and closed choice on the naturals

A function $d\colon M \times M \to \mathbb{R}$ is called a **pseudo-metric** on a set M if it is positive, symmetric and fulfills $d(x,x) = 0$ and the triangle inequality

$$d(x,z) \leq d(x,y) + d(y,z).$$

It is called a **metric** if $d(x,y) = 0$ implies $x = y$. A pair (M,d) is called a **pseudo-metric space** if d is a pseudo-metric on M and a **metric space** if d is a metric. Every pseudo-metric space comes with a topology that is generated by the open balls with respect to the pseudo-metric and therefore with notions of continuity of functions between and limits of sequences in pseudo-metric spaces. The latter is of particular importance since any pseudo-metric space is first-countable and thus knowing the limits of sequences is sufficient for characterizing continuity. A more accessible definition of continuity can be given using the well-known ε-δ-criterion that does not require any knowledge about topology. A function $f\colon N \to M$ between pseudo-metric spaces (N, d_N) and (M, d_M) is called **continuous in** x if

$$\forall \varepsilon, \exists \delta, \forall y, d_N(x,y) \leq \delta \implies d_M(f(x), f(y)) \leq \varepsilon.$$

The function is called **continuous** if it is continuous in any point of M. An element x of a pseudo-metric or metric space (M,d) is said to be the **limit** of a sequence (x_n) in M, in symbols $\lim_{(M,d)}(x_n) = x$, if

$$\forall \varepsilon, \exists N, \forall n, N \leq n \implies d(x, x_n) \leq \varepsilon,$$

and a function f between pseudo-metric spaces is said to be **sequentially continuous** if $\lim_{(N,d_N)}(x_n) = x$ implies $\lim_{(M,d_M)}(f(x_n)) = f(x)$.

The notion of a metric captures the properties that one expects a notion of distance to have in a very general fashion. Metric spaces are a widely applicable tool for talking about continuity on many spaces of practical interest and are therefore a common concept in many branches of mathematics. As such, metric spaces have received considerable attention in their formal treatment. In particular there exists a definition of the concept of a metric space and continuity of functions between metric spaces in the standard library of Coq.

Several external libraries come with their own versions of metric spaces and continuity. Metric spaces and uniformly continuous functions are some of the core concepts of the C-CoRn library and developed in a way that avoids mentioning real numbers [O'C09]. Another example is the Coquelicot library [BLM15], a widely used conservative extension of the classical formalization of the real numbers provided by Coq's standard library. Coquelicot uses a concept it refers to as uniform space, but which is actually more restrictive than the mathematical notion of a uniform space and closely resembles pseudo-metric spaces (`cntp_cntp`). The definitions of limits and continuity used in Coquelicot rely on filters instead of sequences. Most of these design choices are not arbitrary but for good reasons. The avoidance of real numbers in the treatment of metric spaces in C-CoRn is due to the existence of different real number objects in a fully constructive setting. For Coquelicot the choice of pseudo-metric spaces over metric spaces is due to neither Coq, nor the axioms of the real numbers, proving functional extensionality. This makes it challenging to define a metric on any kind of space of functions. A pseudo-metric can often be defined in a straight forward manner. As the "uniform spaces" are Coquelicot's most general structure and are first-countable, the definitions via filters are equivalent to those using sequences, however, it is not clear whether equivalence of derived concepts can always be proven in the restricted setting that Coquelicot works in.

The Incone library comes with its own version of metric spaces that is kept close to the classical mathematical treatment and is thus most similar to the metric spaces that can be found in Coq's standard library. It provides interfaces with both the standard library of Coq (`MS2M_S`, `M_S2MS`, `Uncv_lim`, `cont_limin`, etc.) and the Coquelicot library (`US2MS`, `MS2US`, `cntp_cntp`, etc.) so that it is possible to reuse results proven there (the lemmas `limD`, `limM`, `R_cmplt` and many more are proven this way). In contrast to the Coquelicot library, the metric library does not attempt to be conservative over the background theory of the real numbers. The main advantage of working constructively is the ability to extract computational content and this ability is lost as soon as the real numbers from the standard library enter the stage. The metric library further diverges from Coquelicot by using sequences instead of filters. This makes the comparison to represented spaces easier and is more appropriate for easy accessibility of computational content.

For an easy back and forth between convergence statements quantifying over real numbers and versions that quantify over integers instead, a line of lemmas is provided that contain the phrase "tpmn" (for "two to the power minus n") in their name (`tpmnP`, `lim_tpm`, `dns_tpmn`, etc.). For instance `tpmnP` proves that the propositional $2^{-n} \leq 2^{-m}$ on real numbers reflects the boolean $m \leq n$ on the math-comp natural numbers and `lim_tpmn` says that in the definition of the limit one may replace ε by 2^{-n} and thereby quantification over $\mathbb{R}$ by quantification over $\mathbb{N}$. This reduces the descriptive complexity quite a bit for the price of introducing an additional type. For the rational numbers a similar set of lemmas and additionally a constructive instantiation of the restriction of the up function is provided (`upQ`, `limQ`, `archimedQ`, etc.). The rational up function is useful for recovering computational content from proofs in the standard library, as it can often be supplemented for the up function that cannot be defined as a function on a constructive instantiation of the real numbers.

While the naming of notions for metric spaces is identical to what we used for represented spaces, there are some conceptual differences. First off, it is well known that a function between metric spaces is continuous if and only if it is sequentially continuous, where for represented spaces the backward implications can fail and admissibility of the involved representations is a sufficient condition to recover it. Secondly, in the case of metric spaces both continuity and its sequential variant can be recovered from point-wise such notions while for represented spaces this is only the case for sequential continuity. Indeed, the point-wise notions introduce subtle problems in the treatment of sub-spaces. Even in the most well-behaved cases as for a closed interval as a subspace of the real numbers there is a difference between a function on the reals being continuous in each point of the interval and the restriction of the function to the interval being continuous. The characteristic function of the interval has a continuous restriction but is not continuous in either end-point. This leads the statements of important theorems from the standard library (for instance the mean value theorem) to be slightly off as compared to what a mathematician would expect. We choose to assume proof-irrelevance which allows for a treatment of sub-spaces as dependent types. It should be noted that this comes with its own problems and inconveniences.

$$
\begin{tikzcd}[column sep=huge, row sep=large]
\text{metric spaces} & \text{represented spaces} \\
\begin{array}{c}\text{sequentially}\\\text{continuous}\end{array} \arrow[r, Rightarrow, shift left, "\text{non-constructive}"] \arrow[d, Rightarrow, shift left, "\text{non-constructive}"] & \begin{array}{c}\text{sequentially}\\\text{continuous}\end{array} \arrow[l, Rightarrow, shift left, "\text{non-constructive}"] \\
\varepsilon\text{-}\delta\text{ continuous} \arrow[r, Rightarrow, shift left, "\text{non-constructive}"] \arrow[u, Rightarrow, shift left, "\text{constructive}"] & \begin{array}{c}\text{continuously}\\\text{realizable}\end{array} \arrow[l, Rightarrow, shift left, "\text{constructive}"] \arrow[u, Rightarrow, "\text{constructive}"']
\end{tikzcd}
$$

Figure 5: Implications between different notions of continuity on metric spaces

4.1 Recovering continuity on Baire space from a metric structure

Let $\mathcal{B} = \mathbf{Q} \to \mathbf{A}$ be a Baire space. Since $\mathbf{Q}$ is countable there exists a surjective function $\mathrm{cnt}\colon \mathbb{N} \to \mathbf{Q}$. For each such function cnt define a mapping $d_{\mathrm{cnt}}\colon \mathcal{B} \times \mathcal{B} \to \mathbb{R}$ (`baire_distance` in the library) by

$$d_{\mathrm{cnt}}(\varphi, \psi) := \begin{cases} 2^{-k} & \text{if } \varphi \neq \psi \text{ and } k = \min\{n, \varphi(\mathrm{cnt}(n)) \neq \psi(\mathrm{cnt}(n))\} \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 29 (`dst_pos`, `dst_sym`, `dstxx`, `dst_trngl`, `dst_eq`) *$(\mathcal{B}, d_{\mathrm{cnt}})$ is a metric space.*

For the proof a function named `search` was implemented, this function is a version of the `find` function from the standard library but acts on lists of natural numbers instead of arbitrary lists and thus it is possible to prove lemmas that are not meaningful in general. This function was used earlier: It is what is behind the implementation of the search operator from Example 4. It may have further applications but it is probable that better implementations can be found in other places.

Theorem 30 (`lim_lim`) *Whenever* $\mathrm{cnt}\colon \mathbb{N} \to \mathbf{Q}$ *is surjective, then* $\lim_{(\mathcal{B}, d_{\mathrm{cnt}})} = \lim_{\mathcal{B}}$.

From this theorems, whose proof is very straight-forward, the following can be easily deduced:

Corollary 31 *An function between Baire spaces is sequentially continuous in the sense of Section 2.2 if and only if it is sequentially continuous as function between the metric spaces.*

A stronger version of this corollary for partial operators can be proven, we omit it here and proceed to formulate a theorem about the the non-sequential version of continuity instead.

Theorem 32 (`cont_cont`) *Whenever $\mathcal{B}$ and $\mathcal{B}'$ are Baire spaces and* cnt *and* cnt$'$ *are appropriate surjective functions then $F\colon \subseteq \mathcal{B} \to \mathcal{B}'$ is continuous in the sense of Section 2.2 if and only if it is continuous as function from* $(\mathrm{dom}(F), d_{\mathrm{cnt}})$ *to* $(\mathcal{B}', d_{\mathrm{cnt}'})$.

Just like for the previous theorem, the proof is very straightforward and a special case was proven as an example before the metric space structure on Baire-space was available.

Example 33 (`examples/continuous_search.v`) The regular notion of continuity on the original Baire space $\mathbb{N}^{\mathbb{N}}$ is captured by the continuity introduced in Section 2.2 if all of the types are substituted with the natural numbers.

4.2 Comparing continuity in represented and in metric spaces

Metric spaces are well investigated in computable analysis [Wei93]. In particular in the case where (M, d) is a metric space and (r_n) is a designated dense sequence in M, it is well known that the multifunction $\delta_{\mathbf{M}}$ defined by

$$x \in \delta_{\mathbf{M}}(\varphi) \quad \iff \quad \forall n, d(x, r_{\varphi(n)}) \leq 2^{-n}.$$

defines a representation $\delta_{\mathbf{M}}$ of M (`mrep_sing` and `mrep_sur`) and we denote the corresponding represented space by $\mathbf{M} := (M, \delta_{\mathbf{M}})$. Note that this representation is very close to how real numbers where represented: A name of an element of a metric space produces an index of an approximation from an accuracy requirement. It is also well known that in this case it is true that a sequence (x_n) in M converges to a limit x from M with respect to the metric space structure if and only if they converge as elements of the represented space $\mathbf{M}$, or for short $\lim_{(M,d)} = \lim_{\mathbf{M}}$. Furthermore, if (M', d') is another metric space with dense sequence (r'_n), then a function $f : M \to M'$ is continuous as a function between metric spaces if and only if it is continuously realizable as a function $f : \mathbf{M} \to \mathbf{M}'$.

The above fixes the query and answer types to be $\mathbb{N}$. Since a query about an element of any metric space is a precision requirement this is reasonable. For the answer type it would be better to be more general and in the formulation. While the Incone library does currently not do that, we plan to soon allow any countable and inhabited type. The reason for this is that for metric spaces like the continuous functions on the unit interval $C([0, 1])$ with the metric induced by the supremum norm, candidates for dense sub-sequences are the rational polynomials. Thus it is desirable to use the Mathematical Components type directly instead of taking the detour of enumerating the rational polynomials.

This section describes our formal proofs about the comparisons of the two continuity notions and with their sequential versions (compare Figure 5). The proofs have been kept as constructive as possible. Since the definition of a metric space relies on the axiomatic reals, only one of the implications is fully constructive, the others are constructive over the background theory of real numbers and do not rely on the axioms of the real numbers in an essential way. A metric space is called separable if there exists a dense sequence and even though the sequence goes into the definition of the corresponding Cauchy representation, we decide to not mention it explicitly in the following. This is justified in a continuity setting as it is well known that different choices of dense sequences lead to isomorphic represented spaces. As always, the situation is more complicated if computability is considered and in this case one should assume that for the following two metric spaces with dense sequences are fixed.

Let us first describe the proof of the equivalence of the notions of sequential continuity. The main part of the proof is that the notions of limit in the metric space and the corresponding represented space coincide.

Theorem 34 (`lim_mlim`) *Whenever (M, d) is a separable metric space and $\mathbf{M}$ as above then*

$$\lim_{(M,d)} = \lim_{\mathbf{M}}.$$

The proof that the convergence in the represented space implies the convergence in the metric space is straight forward. The idea behind the other direction can be sketched as follows: If (x_n) converges to x in the metric space then there exists a modulus of convergence, i.e. some $\mu : \mathbb{N} \to \mathbb{N}$ such that

$$\forall n, m \in \mathbb{N}, m \geq \mu(n) \Rightarrow d(x_m, x) \leq 2^{-n}.$$

From an arbitrary sequence (φ'_m) of names of the x_m and a name φ' of x an appropriate convergent sequence of names can be defined by

$$\varphi_m(n) := \begin{cases} \varphi'(n+1), & \text{if } \mu(n+1) \leq m \\ \varphi'_m(n), & \text{otherwise} \end{cases}$$

and its limit is given by $\varphi(n) := \varphi'(n+1)$ which is clearly a name of x again. It should be clear from this sketch that some rather weak choice principles have to be assumed to get hold of μ. This could probably be eliminated by appropriate assumptions about the values of the metric being approximable on the elements of the dense sequence (i.e. by working with computable metric spaces).

The proof that the sequential notions of continuity on metric and represented space coincide follows immediately from this theorem. As the proof of each direction requires to translate limits in both directions, either of the directions is as constructive or non-constructive as the worse direction of the previous theorem.

Corollary 35 (`scnt_mscnt`) *If (M, d) and (M', d') are separable metric spaces, then a function $f : M \to N$, is sequentially continuous as a function between metric spaces if and only if it is sequentially continuous as function $f : \mathbf{M} \to \mathbf{M}'$.*

For the equivalence of ε-δ-continuity and continuous realizability one direction needs stronger assumptions and for the INCONE library we thus separated the proofs.

Lemma 36 (`cont_mcont` **and** `mcont_cont`) *Let (M, d) and (M', d') be two separable metric spaces. A function $f : M \to M'$ is ε-δ-continuous if and only if $f : \mathbf{M} \to \mathbf{M}'$ is continuous.*

While the proof that continuous realizability implies ε-δ-continuity is straight forward, the proof of the other implication required some work and we sketch some of the details.

Interestingly, the tools needed for this proof are in spirit fairly close to those that were used to prove the existence of associates in Section 2.3, more specifically we also use minimal moduli. Call a function $\mu : \mathbb{N} \to \mathbb{N}$ a **metric modulus of continuity** of f in x if

$$\forall y : M, d(x, y) \leq 2^{-\mu(n)} \implies d(f(x), f(y)) \leq 2^{-n}$$

And call such a modulus **minimal** if it is minimal in the obvious way. Note that this notion generalizes the one discussed in Section 2.3 if Baire space is equipped with the metric space structure from 4.1 and reasonable assumptions about the enumeration used for this are made.

Lemma 37 (`exists_minmod_met`) *For any continuous function f between metric spaces and any argument x for f there exists a minimal modulus of f in x.*

As the version for Baire-space is implied by this lemma and we noted that the existence of a minimal modulus of an operator on Baire space cannot be proven without axioms, the proof of this has the classical axiom in its assumptions.

The proof of the lemma `mcont_cont` is different from the Baire space case, as the type of a modulus of continuity has changed and rendered the notion of a self-modulating modulus of continuity meaningless. Indeed, if the metric space is connected, the function assigning to each x the minimal modulus function of f in x cannot be continuous as it takes values in a totally disconnected space. One might expect that this is due to the awkward typing, and that making μ have type $\mathbb{R} \to \mathbb{R}$ instead would help, but it does not. It is known that also in this case the minimal modulus need not be continuous and that a construction of a continuous modulus of continuity, while possible in general, takes considerably more effort []. Thus, our proof that ε-δ-continuity implies continuous realizability uses a notion of being almost-self-modulating instead, where the value of the minimal modulus on slightly disturbed input from the metric space is bounded in terms of a shift of the minimal modulus in the original value.

4.3 Sierpiński space and closed choice on the naturals

This section describes the content of the file `examples/closed_choice.v` from the INCONE library. Sierpiński space $\mathbb{S}$ (`cs_Sirp` in the library) is the space whose base set is the two point set $\{\bot, \top\}$ equipped the total representation $\delta_{\mathbb{S}}$ with names of type $\mathbb{N} \to \mathbb{B}$ specified by

$$\delta_{\mathbb{S}}(\varphi) = \top \quad \iff \quad \exists n \in \mathbb{N}\ \varphi(n) \neq \text{false}.$$

For a subset $U \subseteq \mathbf{X}$ denote by χ_U its characteristic function

$$\chi_U : \mathbf{X} \to \mathbb{S}, \quad \chi_U(x) := \begin{cases} \top & \text{if } x \in U, \\ \bot & \text{otherwise.} \end{cases}$$

The set $U \subseteq \mathbf{X}$ is open if and only if this characteristic function χ_U is continuous as a function from $\mathbf{X}$ to $\mathbb{S}$. For this reason Sierpiński space plays an important role in computable analysis. Following for instance [Pau16] we can identify the space $\mathcal{O}(\mathbf{X})$ of open subsets of $\mathbf{X}$ (`O( _ )` in the library) with the space of continuous functions $\mathbb{S}^{\mathbf{X}}$ from Section 3.3. Similarly, the space $\mathcal{A}(\mathbf{X})$ (`A( _ )` in the library) of closed subsets of $\mathbf{X}$ is represented as the complements of opens.

For concrete spaces $\mathbf{X}$ it is often the case that simpler descriptions of $\mathcal{O}(\mathbf{X})$ and $\mathcal{A}(\mathbf{X})$ are available. If the represented space $\mathbf{X} = \mathbb{N}$ are the natural numbers, for instance, one may make use of the infinite product construction from Section 3.2 and in particular of the special case $I = \mathbb{N}$ and $\mathbf{X} = \mathbb{S}$ of Lemma 28 which guarantees that $\mathcal{O}(\mathbb{N}) = \mathbb{S}^{\mathbb{N}} \simeq \prod_{\mathbb{N}} \mathbb{S} = \mathbb{S}^{\omega}$. There exists a fully concrete description of $\mathcal{O}(\mathbb{N})$ that is often used for reasoning about this space in computable analysis: The enumeration representation, where a name of an open set enumerates its elements. We call the corresponding space $\mathbf{O}_{\mathbb{N}}$ (`O_N` in the library). The representation of the corresponding concrete space $\mathbf{A}_{\mathbb{N}}$ of the closed subsets of the natural numbers (`A_N` in the library) is given by

$$\delta_{\mathbf{A}_{\mathbb{N}}}(\varphi) = \mathbb{N} \setminus \{n \colon \mathbb{N} \mid \exists m \colon \mathbb{N},\ \varphi(m) = n + 1\}.$$

The information a name specifies about a closed set is an enumeration of its complement.

We provide a formal proof that the enumeration representations of the open and closed subsets of the natural numbers capture the abstract structure these spaces through the exponential in the category of represented spaces.

Theorem 38 **(`AN_iso_Anat`, `ON_iso_Onat` and `clsd_iso_open`)** *$\mathcal{A}(\mathbb{N}) \simeq \mathbf{A}_{\mathbb{N}}$, $\mathcal{O}(\mathbb{N}) \simeq \mathbf{O}_{\mathbb{N}}$ and $\mathcal{A}(\mathbb{N}) \simeq \mathcal{O}(\mathbb{N})$.*

The last of these isomorphies is trivial, the isomorphism is taking the complement and it is realized by the identity function. The isomorphism of $\mathcal{O}(\mathbb{N})$ and $\mathbf{O}_{\mathbb{N}}$ is proven by first replacing $\mathcal{O}(\mathbb{N})$ by $\mathbb{S}^{\omega}$ as described above. The realizers for the isomorphisms between $\mathbb{S}^{\omega}$ and $\mathbf{O}_{\mathbb{N}}$ can be defined as functions directly, where one direction uses the Cantor paring function provided by the mathematical components library.

As an application of the above let us consider choice operators. The task $C_{\mathbf{X}}$ of closed choice on $\mathbf{X}$ is embodied by the multivalued function

$$C_{\mathbf{X}} \colon \mathcal{A}(\mathbf{X}) \rightrightarrows \mathbf{X}, \quad a \in C_{\mathbf{X}}(A) \iff a \in A.$$

Or in words: a is an acceptable return value of $C_{\mathbf{X}}$ on input A if and only if a is an element of A. Note that this in particular means that the domain of $C_{\mathbf{X}}$ are the non-empty subsets of $\mathbf{X}$ and that a realizer can behave arbitrarily outside of the domain, i.e., no solution needs to be produced in this case and even divergence is allowed.

Consider the special case were $\mathbf{X} = \mathbb{N}$. While the domain of the multivalued function $C_{\mathbb{N}}$ is $\mathcal{A}(\mathbb{N})$ we may use the same definition to obtain a multifunction $C'_{\mathbb{N}} \colon \mathbf{A}_{\mathbb{N}} \rightrightarrows \mathbb{N}$. A mathematician may even consider it pointless to give this function a different name as isomorphic spaces are regularly identified. Indeed, for the question of whether $C_{\mathbb{N}}$ has a continuous realizer the space $\mathcal{A}(\mathbf{N})$ may be substituted with $\mathbf{A}_{\mathbb{N}}$ for this exact reason.

Corollary 39 **(`CN_CN'_hcr`)** *$C_{\mathbb{N}}$ has a continuous realizer if and only if $C'_{\mathbb{N}}$ does.*

Thus, we may prove that closed choice on the naturals does not have a continuous realizer.

Theorem 40 **(`CN'_not_cont`)** *$C'_{\mathbb{N}}$ does not have a continuous realizer.*

The proof proceeds by contradiction. Assume that to the contrary $C'_{\mathbb{N}}$ is continuous and F is a continuous realizer. Pick any name φ of the one point set $\{0\}$. As F is a realizer, it has to return a name of 0 on input φ, i.e., $F(\varphi)(\star) = 0$. If F is continuous there is a list $L \subseteq \mathbb{N}$ such that $F(\varphi)(\star) = F(\psi)(\star)$ for all $\psi\colon \mathbb{N} \to \mathbb{N}$ that coincide with ϕ on L. Consider the name φ' of the non-empty set $A := \mathbb{N} \setminus (\{n+1 \mid \exists m \in L, \varphi(m) = n\} \cup \{0\})$ defined by

$$\varphi'(n) := \begin{cases} \varphi(n), & \text{if } n \in L \\ 1, & \text{otherwise.} \end{cases}$$

On one hand, $F(\varphi')(\star) \in A$ since F is a realizer. On the other hand $F(\varphi')(\star) = F(\varphi)(\star) = 0$ as φ and φ' coincide on L and $0 \notin A$. This is a contradiction and completes the proof.

From the theorem and the previous corollary the following is immediate:

Corollary 41 (`CN_not_cont`) *Closed choice on the natural numbers is discontinuous.*

5 Conclusion

The emphasis of the incone library is different from that of developments like C-CoRn: It aims to provide general tools for enriching abstract mathematical structures of interest with computational structure, for comparing different such datasets and for providing computational content for mathematical statements or proving this to be impossible. We feel that the examples from this paper showcase these capabilities well and involve many of the most prominent features of Incone: The abstract definition of the space of open subsets is based on the libraries function space construction and our proof of isomorphy relies on infinite products of represented spaces. Many of our examples fall outside of the scope of developments like C-CoRn. Results from C-CoRn could be reformulated in our setting and while it is one of our future goals to provide compatibility it is unclear how much effort translation of content takes and whether this provides additional insight. In our opinion a second access to similar topics with a different focus and an alternate presentation of the contents has a right to exist. We believe the Incone library to be reasonably accessible for the computable analysis community and hope that its combination with methods from that community [MPPZ16] could help to make parts of it more accessible to the numerical analysis community.

The incone library keeps close to recent work about complexity theory for computable analysis [KC12, Fer17, NS17, KST19] such that it should be possible to add capabilities to at least do qualitative complexity theory in terms of tracking the rate of decrease in accuracy of approximations in the future. A full treatment of step-counting complexity might become available in the not too distant future due to recent progress on the formalization of models of computation [FS18, Max18] and methods from implicit complexity theory [FHM+18]. Another way to gain insight into such efficiency considerations would be to capture the trace of the basic feasible functionals on the operators on Baire space [Meh76, KC96, KS18].

There are some gaps in the Incone library that are worth filling. For instance, currently we do not have a complete proof that the category of represented spaces is Cartesian closed as the corresponding universal property has not been proven formally. The library proves a restricted case by providing a duality operator, but a full proof would be desirable. While it is also possible to start proving induction principles, we think that it is not reasonable to do at this point as they are prone for introducing inefficiencies. The problems one is faced with are illustrated by the description of polynomial evaluation in Section 3.1. Our line of reasoning here is that, until we have a better solution, cases where an induction principle is needed that can not be traced back to one on discrete data, are probably points were customization should be done and work needs to be invested to keep algorithms efficient. The reflection using the Rlzrs library may be helpful in understanding which parts of the algorithms from computable analysis introduce inefficiencies and how these can be come by.

The replacement of Baire space by more general spaces means that we maintain the ability to benefit from Coq's machinery in the low-level manipulations of data. From an abstract point of view this makes our approach look like an attempt to interpret a class of generalized Kleene-Kreisel continuous functionals as a computational model in presence of an ambient model of computation. Maybe the best way to look at this is as a backwards approach to the more common idea of identifying a sub-algebra that captures computability in a given partial combinatory algebra, in this case K_2 [LN15, Bau00]. Most of the methods from the Rlzrs library are not original and have been implemented independently of a specific proof assistant before [BS07]. Their combination with a machine based approach would remove the current restriction to only talk about computability on the meta-level and the reliance on Coq specific mechanisms. By implementing it in other proof assistants one could trade convenience in computationally operating on discrete data against bigger mathematical libraries.

We feel that this paper provides sufficient evidence that the concepts developed in the Incone library can be used as a foundation for proving statements from computable analysis in Coq. The possible applications we are interested to look into are manifold. One particularly fitting extension of the contents of this paper would be a proof that $C([0,1]) \simeq \mathbb{R}^{[0,1]}$. This statement is called the Computable Weierstraß Theorem [PEC75]: $C([0,1])$ is represented as separable metric space with supremum norm and the rational polynomials as dense sequence and $\mathbb{R}^{[0,1]}$ is a function space. Other possibilities include:

- A more computation-efficient representation of real numbers and results about ODE solving [IH12, MS13, KST18]. This may be done by providing an interface with C-CoRn, parts of it could also be done separately by relying on libraries like coq-Interval.
- Duality theory for spaces of sumable sequences (ℓ_p-spaces) which provide a pool of examples where sub-spaces of exponentials can be treated complexity theoretically [Sch04, SS17]. Additionally it constitutes a step towards capturing popular methods for solving partial differential equations [BY06, SS08, BCF$^+$17].
- A characterization of continuity via preimages of open sets, general considerations about admissibility, discreteness, compactness and many other similar results [Pau16, Sch02b].

References

[AB14] Jeremy Avigad and Vasco Brattka. *Computability and analysis: the legacy of Alan Turing*, page 1–47. Lecture Notes in Logic. Cambridge University Press, 2014.

[ACR18] Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *Journal of Formalized Reasoning*, 11(1):43–76, 2018.

[ASBZ13] Klaus Ambos-Spies, Ulrike Brandt, and Martin Ziegler. Real benefit of promises and advice. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 1–11, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Bau00] Andrej Bauer. *The Realizability Approach to Computable Analysis and Topology.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2000. AAI3002721.

[BB12] Errett Bishop and Douglas Bridges. *Constructive analysis*, volume 279. Springer Science & Business Media, 2012.

[BCF$^+$13] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program. *Journal of Automated Reasoning*, 50(4):423–456, April 2013.

[BCF+17] Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq Formal Proof of the Lax–Milgram theorem. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 79–89, Paris, France, January 2017. ACM.

[BDBP12] Vasco Brattka, Matthew De Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163(8):986–1008, 2012.

[BG11] Vasco Brattka and Guido Gherardi. Effective choice and boundedness principles in computable analysis. *Bulletin of Symbolic Logic*, 17(1):73–117, 2011.

[BGP17] Vasco Brattka, Guido Gherardi, and Arno Pauly. Weihrauch complexity in computable analysis. *arXiv preprint arXiv:1707.03202*, 2017.

[BKMP16] Vasco Brattka, Akitoshi Kawamura, Alberto Marcone, and Arno Pauly. Measuring the Complexity of Computational Content (Dagstuhl Seminar 15392). *Dagstuhl Reports*, 5(9):77–104, 2016.

[Bla01] Jens Blanck. Exact real arithmetic systems: Results of competition. In *Computability and complexity in analysis. 4th international workshop, CCA 2000. Swansea, GB, September 17–19, 2000. Selected papers*, pages 389–393. Berlin: Springer, 2001.

[BLM15] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015.

[BLM16] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016.

[BM17a] Sylvie Boldo and Guillaume Melquiond. *Computer Arithmetic and Formal Proofs.* ISTE Press - Elsevier, December 2017.

[BM17b] Sylvie Boldo and Guillaume Melquiond. *Computer Arithmetic and Formal Proofs.* ISTE Press – Elsevier, 2017.

[BRT17] Yves Bertot, Laurence Rideau, and Laurent Théry. Distant decimals of π. *Journal of Automated Reasoning*, pages 1–45, 2017.

[BS07] Andrej Bauer and C.A. Stone. RZ: A tool for bringing constructive and computable mathematics closer to programming practice. *Computation and Logic in the Real World. CiE 2007. Lecture Notes in Computer Science, vol 4497.*, 2007.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

[BY06] Vasco Brattka and Atsushi Yoshikawa. Towards computability of elliptic boundary value problems in variational formulation. *Journal of Complexity*, 22(6):858 – 880, 2006. Computability and Complexity in Analysis.

[CFGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.

[Coh12] Cyril Cohen. Construction of real algebraic numbers in Coq. In Lennart Beringer and Amy Felty, editors, *ITP - 3rd International Conference on Interactive Theorem Proving - 2012*, Princeton, United States, August 2012. Springer.

[Coh13] Cyril Cohen. Pragmatic quotient types in coq. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 213–228, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Dei92] K. Deimling. *Multivalued Differential Equations*. De Gruyter series in nonlinear analysis and applications. W. de Gruyter, 1992.

[EM46] Samuel Eilenberg and Deane Montgomery. Fixed point theorems for multi-valued transformations. *American Journal of Mathematics*, 68(2):214–222, 1946.

[EX16] Martín Escardó and Chuangjie Xu. A constructive manifestation of the Kleene–Kreisel continuous functionals. *Annals of Pure and Applied Logic*, 167(9):770 – 793, 2016. Fourth Workshop on Formal Topology (4WFTop).

[Fer17] Hugo Feree. Game semantics approach to higher-order complexity. *J. Comput. Syst. Sci.*, 87(C):1–15, August 2017.

[FHM+18] Hugo Férée, Samuel Hym, Micaela Mayero, Jean-Yves Moyen, and David Nowak. Formal proof of polynomial-time complexity with quasi-interpretations. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 146–157. ACM, 2018.

[FS18] Yannick Forster and Gert Smolka. Call-by-Value Lambda Calculus as a Model of Computation in Coq. *Journal of Automated Reasoning*, 2018.

[Grz57] A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–71, 1957.

[IH12] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In *ITP*, volume 7406 of *LNCS*, pages 377–392, 2012.

[KC96] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25:117–132, 1996.

[KC12] Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Transactions in Computation Theory*, 4(2):Article 5, 2012.

[Kle59] S.C. Kleene. Countable functionals. *Constructivity in Mathematics: proceedings of the colloquium held at Amsterdam*, 1959.

[KN17] Michal Konecný and Eike Neumann. Representations and evaluation strategies for feasibly approximable functions. *CoRR*, abs/1710.03702, 2017.

[Ko91] Ker-I Ko. *Complexity theory of real functions*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1991.

[KP14] Akitoshi Kawamura and Arno Pauly. Function spaces for second-order polynomial time. In *Conference on Computability in Europe*, pages 245–254. Springer, 2014.

[Kre59] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite type, Constructivity in Mathematics, 1959.

[KS18] Bruce M. Kapron and Florian Steinberg. Type-two polynomial-time and restricted lookahead. In *LICS*, 2018.

[KST18] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ODE solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.

[KST19] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Second-order linear-time computability with applications in computable analysis. *15th Annual Conference on Theory and Applications of Models of Computation*, 2019. extended abstract accepted for presentation at TAMC 2019.

[KW85] Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical computer science*, 38:35–53, 1985.

[Lac58] Daniel Lacombe. Sur les possibilités d'extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles. In *Le raisonnement en mathématiques et en sciences expérimentales*, Colloques Internationaux du Centre National de la Recherche Scientifique, LXX, pages 67–75. Editions du Centre National de la Recherche Scientifique, Paris, 1958.

[LN15] John Longley and Dag Normann. *Higher-order computability*, volume 100. Springer, 2015.

[LR08] Robert S. Lubarsky and Michael Rathjen. On the constructive dedekind reals. *Logic and Analysis*, 1(2):131–152, May 2008.

[mat] The Mathematical Components library. `https://math-comp.github.io/math-comp/`.

[Max18] MaximilianWuttke. Verified programming of Turing Machines in Coq. Master's thesis, Saarland University, 2018.

[Meh76] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Sciences*, 12(2):147 – 178, 1976.

[MPPZ16] Norbert Th. Müller, Sewon Park, Norbert Preining, and Martin Ziegler. On formal verification in imperative multivalued programming over continuous data types. *CoRR*, abs/1608.05787, 2016.

[MS13] Evgeny Makarov and Bas Spitters. The Picard algorithm for ordinary differential equations in Coq. In *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 463–468. Springer, 2013.

[Mül01] Norbert Th. Müller. The iRRAM: Exact arithmetic in C++. In *Computability and complexity in analysis. 4th international workshop, CCA 2000. Swansea, GB, September 17–19, 2000. Selected papers*, pages 222–252. Berlin: Springer, 2001.

[NS17] Eike Neumann and Florian Steinberg. Parametrised second-order complexity theory with applications to the study of interval computation. *CoRR*, abs/1711.10530, 2017. submitted for publication.

[O'C05] Russell O'Connor. Essential incompleteness of arithmetic verified by Coq. In Joe Hurd and Tom Melham, editors, *Theorem Proving in Higher Order Logics*, pages 245–260, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[O'C09] Russell O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* PhD thesis, Radboud Universiteit Nijmegen, 2009.

[Pau12] Arno Pauly. Multi-valued functions in computability theory. In *Conference on Computability in Europe*, pages 571–580. Springer, 2012.

[Pau16] Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2016.

[PEC75] Marian Boykan Pour-El and Jerome Caldwell. On a simple definition of computable function of a real variable-with applications to functions of a complex variable. *Mathematical Logic Quarterly*, 21(1):1–19, 1975.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*, volume Volume 1 of *Perspectives in Mathematical Logic*. Springer-Verlag, Berlin, 1989.

[PS18] Arno Pauly and Florian Steinberg. Comparing representations for function spaces in computable analysis. *Theory of Computing Systems*, 62(3):557–582, 2018.

[PZ13] Arno Pauly and Martin Ziegler. Relative computability and uniform continuity of relations. *J. Logic & Analysis*, 5, 2013.

[Sch02a] Matthias Schröder. Extended admissibility. *Theoretical computer science*, 284(2):519–538, 2002.

[Sch02b] Matthias Schröeder. *Admissible Representations for Continuous Computations.* PhD thesis, FernUniversität Hagen, 2002.

[Sch04] Matthias Schröder. Spaces allowing type-2 complexity theory revisited. *Math. Log. Q.*, 50:443–459, 09 2004.

[Sel94] Alan L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357 – 381, 1994.

[Soa78] Robert I Soare. Recursively enumerable sets and degrees. *Bulletin of the American Mathematical Society*, 84(6):1149–1181, 1978.

[SS08] Svetlana Selivanova and Victor Selivanov. Computing solutions of symmetric hyperbolic systems of pde's. *Electron. Notes Theor. Comput. Sci.*, 221:243–255, December 2008.

[SS17] Matthias Schröder and Florian Steinberg. Bounded time computation on metric spaces and Banach spaces. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[Ste17] Florian Steinberg. *Computational Complexity Theory for Advanced Function Spaces in Analysis.* PhD thesis, Technische Universität, 2017.

[Ste19a] Florian Steinberg. The CoqRep library. `https://github.com/FlorianSteinberg/coqrep`, 2019.

[Ste19b] Florian Steinberg. The Incone library. `https://github.com/FlorianSteinberg/incone`, 2019. release v1.0.

[Ste19c] Florian Steinberg. The Metric library. `https://github.com/FlorianSteinberg/metric`, 2019. release v1.0.

[Ste19d] Florian Steinberg. The Mf library. `https://github.com/FlorianSteinberg/mf`, 2019. release v1.0.

[Ste19e] Florian Steinberg. The Rlzrs library. `https://github.com/FlorianSteinberg/rlzrs`, 2019. release v1.0.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1936.

[Tur38] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 2(1):544–546, 1938.

[TvD88] A. S. Troelstra and D. van Dalen. *Constructivism in mathematics. Vol. II*, volume 123 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988.

[Wei93] Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113(2):191–210, 1993.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin/Heidelberg, 2000.